## vol. II programação

Para quem já se "alfabetizou" no volume I, este livro ensina as técnicas de programação, desenvolvendo os programas passo a passo a partir de problemas concretos.

Um excelente livro-texto para escolas e para o usuário de um computador pessoal da linha SINCLAIR.

PIERLUIGI PIAZZI, físico, químico e professor há 25 anos.

FLAVIO ROSSINI, engenheiro especialista em automação e transmissão de dados por fibras ópticas.

Encontram-se pela primeira vez no ANGLO vestibulares em 1976 como professor e aluno, discutindo ENCONTRO COM RAMA de ARTHUR C. CLARKE (existe força centrífuga?).

Em 1982, FLAVIO, já lidando com linguagem de computação, aprende o BASIC SINCLAIR com PIERLUIGI num curso intensivo (27 minutos) e se torna professor do NÚCLEO DE ORIENTAÇÃO DE ESTUDOS, cujas apostilas acabaram gerando uma série de livros de sucesso.

Da colaboração dos dois surgiu o BASIC TK, do qual este livro é um dos 3 volumes já de há muito pedido pelos leitores, e muitas vezes adiado por um longo trabalho do FLAVIO nos Estados Unidos.

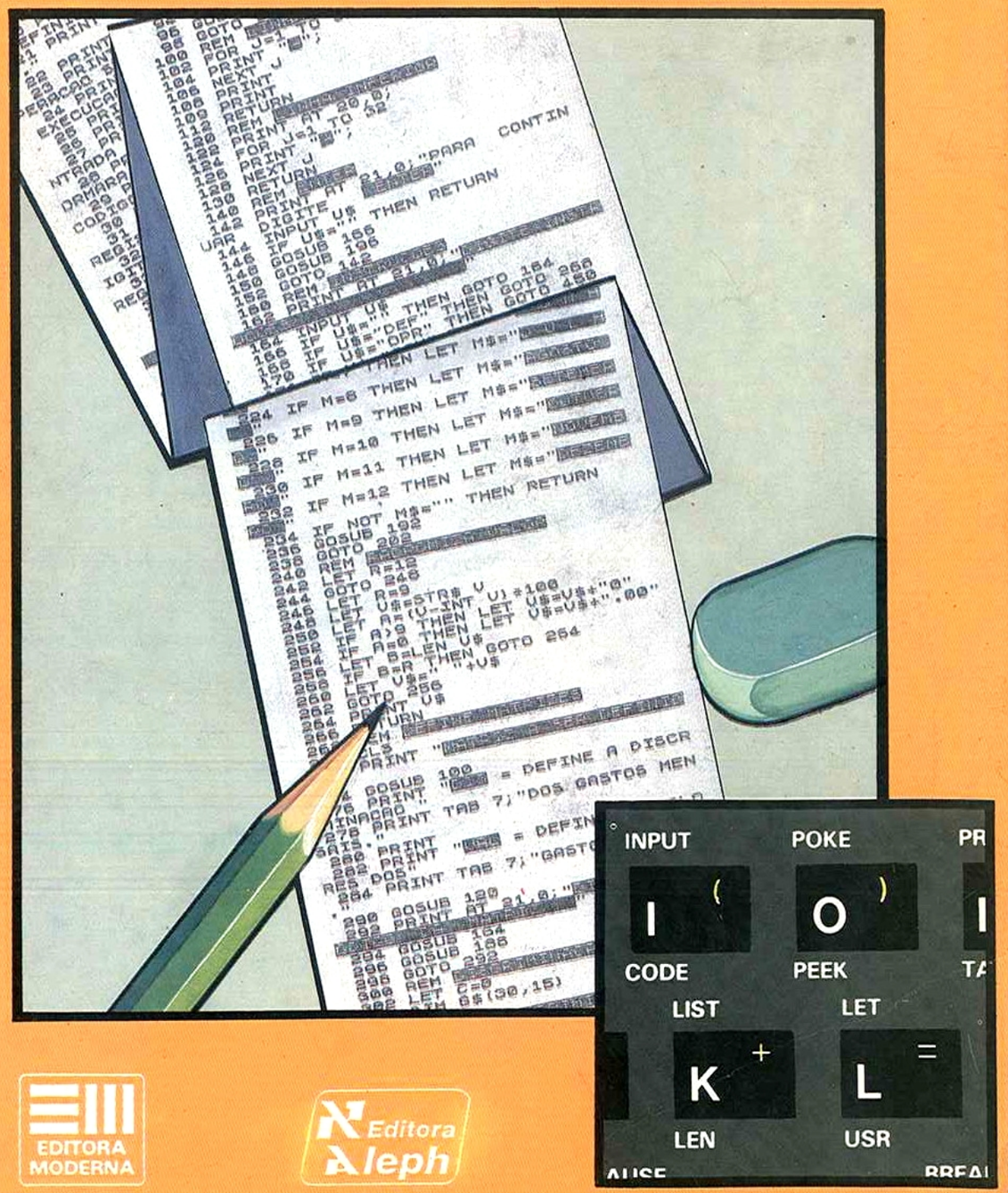
Desta colaboração mais um resultado pronto e uma certeza: força centrífuga **não** existe!

BASIC TK vol. II programação piazzi-rossini

# BASIC TK

## TK 82 83 85 RINGO CP 200
## SINCLAIR ZX 81 TS 1000

vol. II programação piazzi-rossini




EDITORA MODERNA

Editora Aleph

# BASIC TK

## vol. II

## programação

pierluigi piazzi
flavio rossini

# BASIC TK

## vol. II

## programação

Editora Aleph

EDITORA MODERNA

**Coordenação Editorial:**
Betty Fromer Piazzi
**Avaliação, editoração e revisão técnica:**
Nancy Mitie Ariga
**Ilustração da Capa:**
Fátima M. Rossini Gouveia
**Organização Editorial:**
Rosana de Angelo
**Revisão e Copydesk**
Lúcia Kairovsky
**Produção:**
Rosa Kogan Fromer

**Todos os direitos reservados**

Editora Aleph

ALEPH PUBLICAÇÕES E
ASSESSORIA PEDAGÓGICA LTDA.
Av. Brig. Faria Lima, 1451 - conj. 31
01451 - São Paulo - SP
tel.: (011) 813-4555

**Distribuição exclusiva em livrarias**

EDITORA MODERNA

EDITORA MODERNA
R. Afonso Brás, 431
04511 - São Paulo - SP
tel.: (011) 531-5099

1985
1.ª edição
Brasil

**CIP-Brasil. Catalogação-na-Publicação**
**Câmara Brasileira do Livro, SP**

P647c
v.2

Piazzi, Pierluigi, 1943—
Curso de BASIC-TK / Pierluigi Piazzi, Flavio Rossini. —
São Paulo : Aleph : Ed. Moderna, 1985.

1. BASIC (Linguagem de programação para computadores)
2. Microcomputadores — Programação 3. TK (Computador) —
Programação I. Título.

17. CDD-651.8
18. -001.6424
18. -001.642

85-0571

**Índices para catálogo sistemático:**

1. BASIC: Linguagem de programação : Computadores : Processamento de dados 651.8 (17.) 001.6424 (18.)
2. Microcomputadores : Linguagem de programação : Processamento de dados 651.8 (17.) 001.642 (18.)
3. Microcomputadores : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)
4. Programação : Microcomputadores : Processamento de dados 651.8 (17.) 001.642 (18.)
5. TK : Computadores : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)

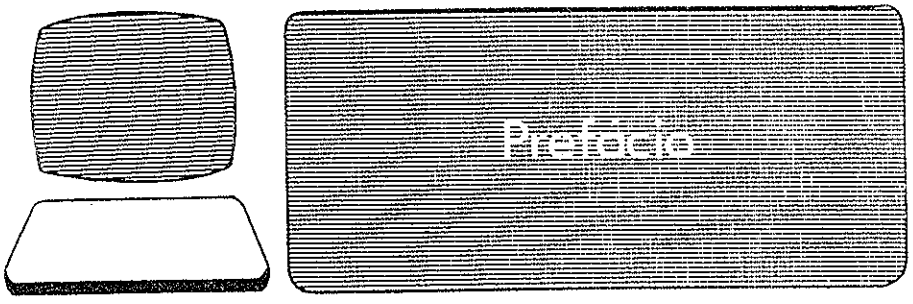
# ÍNDICE

*AGRADECIMENTOS:*

*À Betty e Walleria pelo carinho e incentivo.*

*A Fabio Rendelucci e Marcos Guedes Pereira, pela ajuda na organização e aperfeiçoamento dos programas.*

*A Ricardo F. de Almeida pela verificação do funcionamento do material.*




Uma das leituras que, na infância, mais me aproximou da matemática foi a do livro "O Homem Que Calculava", de Malba Tahan (pseudônimo do escritor e matemático carioca Júlio César de Melo e Sousa, falecido em 1974). Perante a aridez acomodada de um ensino que a transformava em pesadelo, esta obra trouxe-me a surpresa de descobrir o prazer lúdico que a matemática pode proporcionar.

Nesse livro, ante o questionamento de Beremiz (o "homem que calculava") acerca da serventia de seus dotes matemáticos, o narrador respondia-lhe:

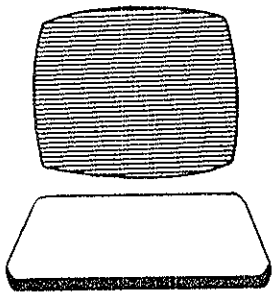
"— A vossa admirável habilidade pode ser empregada em vinte mil casos diferentes. Numa grande capital, como Constantinopla, ou mesmo Bagdá, sereis auxiliar precioso para o governo. Podereis calcular populações, exércitos e rebanhos. Fácil vos será avaliar os recursos do país, o valor das colheitas, os impostos, as mercadorias e todos os recursos do Estado."

Não parece que o autor se refere às possibilidades do computador, esta fascinante "máquina que calcula"?

Ao fazer esta comparação e ler este novo livro de Pierluigi Piazzi e Flavio Rossini, não posso deixar de também comparar o trabalho deles com o de Malba Tahan: a clareza dos exemplos, o raciocínio claro e detalhadamente descrito passo a passo, o estilo fácil e "gostoso" de se ler.

"BASIC TK — volume 2" pressupõe, é claro, a leitura do volume 1 ou conhecimento equivalente, mas nem por isso mergulhou no abismo da complexidade: com o desenvolvimento de um único exemplo (o das "cestas de Natal") Pierluigi e Flavio nos levam a uma verdadeira "viagem ao mundo da programação".

*Carlos Seabra*



# Introdução

Este volume é a continuação do BASIC-TK volume 1, no qual abordamos todos os comandos e funções dos computadores compatíveis com o SINCLAIR ZX-81. No Brasil temos o TK 82-C, o NEZ-8000, o TK-83, o CP-200, o AS-1000 e com algumas diferenças o TK-85, o RINGO R-470 e o CP-200 speed.

No curso de BASIC I, nossa preocupação foi a de "alfabetizar" os alunos em BASIC, possibilitando a elaboração de programas relativamente simples.

No BASIC II, nosso objetivo é possibilitar ao aluno, através de novos conceitos de programação, a elaboração de programas complexos, e acostumá-lo com os **métodos de programação.**

Nas aulas deste curso, iremos ensinar como transformar seus problemas em soluções no computador e mais, como combinar programas simples formando um programa complexo e de grande utilidade.

Abordaremos um problema concreto ("Cestas de Natal") que será usado como exemplo para o desenvolvimento de um programa em BASIC com razoável nível de complexidade.

Em alguns capítulos serão propostos exercícios, cuja resposta está no fim do capítulo correspondente. Aconselhamos ao leitor tentar resolvê-los, pois isto será fundamental para o bom entendimento do texto.

O volume inteiro foi elaborado partindo-se do pressuposto que o leitor domine o BASIC-TK apresentado no volume 1. Havendo dúvidas, ou querendo recordar, aconselhamos consulta ao apêndice A onde resumimos todos os comandos e funções do teclado.

Tanto o volume 1, quanto o volume 2, foram baseados nas apostilas dos cursos BASIC-1 e BASIC-2 que os autores ministram no Núcleo de Orientação de Estudos, sendo fundamentadas, portanto, numa longa experiência didática e tendo passado pelo crivo de centenas de alunos. Esta experiência mostra que é fundamental, para um bom aprendizado, o uso constante e **imediato** do computador para que haja uma interação contínua entre o texto, o aluno e a máquina.

Aconselhamos o leitor, portanto, a munir-se de um micro, papel, lápis, (borracha!), um bom gravador bem ajustado para seu micro e muita vontade de aprender!




Ao desenvolver um programa com um certo grau de complexidade (presente obrigatoriamente em qualquer problema real) é necessário **organizar-se** e **auto-disciplinar-se** para que o objetivo seja atingido no menor tempo e da melhor maneira possível. Assim, existe um raciocínio quase "padrão" a ser adotado pelo programador que embora à primeira vista pareça desnecessário, deve ser seguido para evitar três grandes problemas muito comuns aos programadores "indisciplinados":

- enorme perda de tempo na elaboração e teste do programa e/ou na localização dos erros;
- necessidade de "recomeçar tudo de novo" pela impossibilidade de saber o que está acontecendo;
- impossibilidade de entender o programa no futuro (caso se consiga terminar o programa), quando forem necessárias modificações, expansões, aprimoramentos ou simplesmente para aproveitar um raciocínio feito anteriormente (especialmente se isto for feito por uma pessoa diferente da que elaborou inicialmente o programa).

É aconselhável, portanto, que seja vencida a tentação de "sair programando de cara" e sejam seguidas certas etapas ou estágios que podem inclusive ser imaginadas como um fluxograma que a CPU do programador (ou seja, seu cérebro...) deve seguir. Os estágios a serem seguidos são:

### 1. Definição do problema e desmembramento em blocos principais

Conhecendo-se o "jeito de raciocinar" do computador (ou melhor, da linguagem utilizada) e o problema a ser resolvido,

procura-se formulá-lo nos termos do computador dividindo-se a tarefa a ser realizada em blocos menores, cada qual sendo praticamente um programa independente.

A elaboração de blocos separados e independentes além de facilitar o raciocínio, possibilita a utilização dos mesmos blocos em programas diferentes.

Note que nessa fase **nada** é feito na linguagem do computador. Trata-se apenas de um esquema de raciocínio a ser adotado, onde muitas vezes a resolução "manual" do problema em menores dimensões, facilita a colocação do mesmo nos termos da linguagem.

## 2. Elaboração do fluxograma e listas das variáveis e das flags

Nesta fase, elabora-se o fluxograma (ou diagrama de blocos) para cada um dos blocos definidos na fase anterior (veja apêndice B para recordar fluxogramas).

Durante este procedimento, serão definidas as variáveis a serem utilizadas e as flags (veja vol. 1, cap. 7). É aconselhável, portanto, fazer uma lista das variáveis e flags a serem utilizadas por cada bloco, onde deverá constar o **nome** da variável ou flag, a sua **função** e, no caso das variáveis, especificar quais são comuns a mais do que um bloco (naturalmente dizendo todos os blocos nos quais a variável participa).

## 3. Programação

Agora, "traduz-se" os diagramas de blocos para a linguagem a ser utilizada (no caso o BASIC). Note que o fluxograma além de possibilitar uma visualização melhor do programa, não contém uma série de detalhes (ou refinamentos) que são feitos ao escrever o programa em BASIC. Assim, durante a programação, três pontos devem ser analisados para facilitar o entendimento da listagem no futuro, além do bom andamento do programa:

- colocação de comentários, desde que haja espaço suficiente na memória. Caso não haja memória suficiente, deve-se "tirar" uma listagem do programa na impressora, colá-la numa folha de papel e colocar os comentários ao lado;
- formatação conveniente da saída na tela e na impressora. Todos os casos devem ser analisados para evitar-se paradas do tipo "acabou o espaço na tela";



- "idiot-proof" — normalmente o programa será utilizado por pessoas diferentes daquela que programou o mesmo. Assim, não se sabendo quem irá utilizá-lo, deve ser dedicada muita atenção com relação à colocação de **mensagens auto-explicativas** para o uso do mesmo e muita imaginação deve ser utilizada para **prever todos os tipos de erros** possíveis de serem feitos pelo usuário, evitando a parada do programa (se possível, ao ser feito um erro, o programa deve inclusive dizer o que deve ser feito e/ou qual o tipo de erro cometido).

## 4. Teste, localização e correção de erros (debugging)

Enfrenta-se agora a fase mais "crítica": o programa deve ser rodado em todas as condições possíveis para verificar seu funcionamento (ou seja, se o programa está executando o que deve ser feito). Ocorre, entretanto, que num problema complexo existem muitos tipos de situações e seria praticamente impossível (além de exigir um tempo muito grande) o teste de todas as possibilidades. Desse modo, procura-se fazer o melhor possível, iniciando-se os testes com os casos mais simples, com poucos dados, até chegar-se aos casos mais complexos. Atenção especial deve ser dada às **mensagens, saídas na tela e/ou impressora** e ao **tempo de execução** (que, se necessário, deve ser diminuído ou deve ser colocado na tela um alerta sobre o tempo requerido por determinada parte do programa).

Durante esta fase é de grande valia a ajuda de outra pessoa, pois, por melhor que seja nossa imaginação, existe uma tendência natural de polarizar-se numa determinada linha de raciocínio após terminar um programa, o qual às vezes nos torna "cegos" a certos tipos de erros.

Note agora a importância das flags, especialmente para localizar erros de lógica que obviamente não produzem as mensagens de erro "padrão" do computador.

Um fato muito importante a ser destacado é que uma vez localizado e corrigido um erro, **todos os testes devem ser refeitos**, pois a introdução de qualquer modificação, por menor que seja, pode vir a acarretar erros nos testes que "passaram" antes dela ter sido efetuada.

## 5. Documentação, proteção e acabamento

Uma vez terminado, coloca-se o nome do programa, o nome

do autor e a data. Se possível, deve ser feita uma proteção para evitar os "piratas" de software.

Finalmente, procede-se a uma documentação final, onde deve ser escrito um manual para o usuário, ou seja, as instruções de uso. Devem ser apresentados também os significados das mensagens geradas pelo programa (inclusive as eventuais mensagens de erro), o formato dos dados de entrada e todas as saídas possíveis.

Obviamente, antes de mais nada, deve ser apresentado o propósito do programa e se possível, uma descrição geral da solução adotada.

Para o arquivo do programador, estas instruções devem ser guardadas juntamente com os fluxogramas, listagens e lista de flags e variáveis.

## 6. **Gravação**

Logicamente, o programa deve ser armazenado em algum dispositivo magnético (no nosso caso, uma fita K-7) para possibilitar seu uso futuro e reprodução.

A primeira precaução a ser tomada é fazer uma fita matriz e uma cópia, a qual deve ser guardada num cofre anti-magnético, trancado a sete chaves...

Além disso, outros pontos devem ser analisados:

- não devem ser utilizadas fitas longas, pois elas são delgadas e frágeis, além de produzirem buscas muito demoradas;
- não devem ser utilizadas fitas de muito baixa qualidade;
- o programa deve ser gravado mais do que uma vez na própria fita para diminuir o risco de erro, além de facilitar o LOAD do programa, caso ele não "entre" na primeira vez;
- é aconselhável a gravação com "auto-start" (veja vol. 1, cap. 10), novamente considerando o fato da utilização do programa por outras pessoas;
- de tempos em tempos (aproximadamente um ano), a fita-matriz deve ser refeita, pois há uma natural degradação do sinal pelo uso. É aconselhável também refazer-se a fita-cópia, naturalmente em intervalos de tempo maiores.

Enfim, estes são os seis estágios obrigatórios a serem seguidos na elaboração do programa. Vamos então apresentar o fluxograma a ser seguido pelo programador.



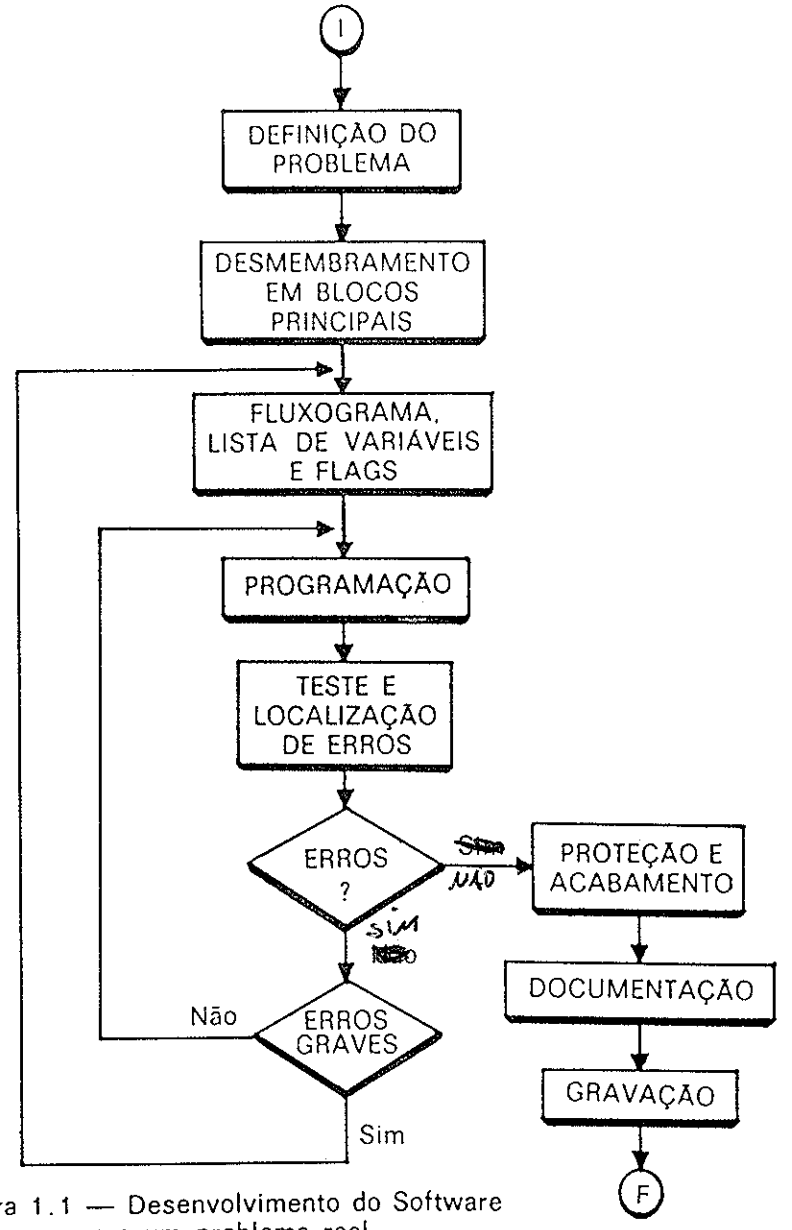


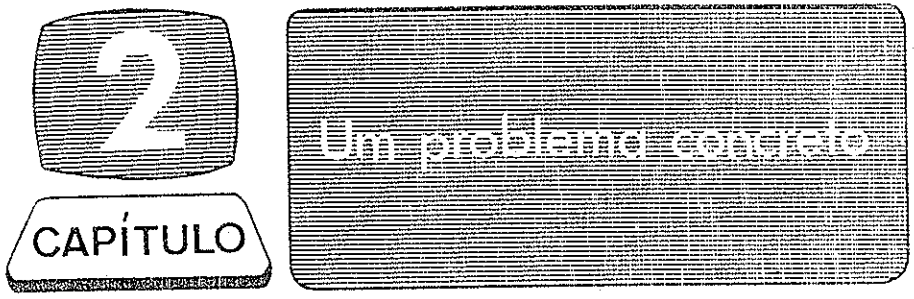

Figura 1.1 — Desenvolvimento do Software para um problema real

Possivelmente, o exato significado dessas fases não tenha ficado completamente esclarecido. No momento, isto não é de muita importância, pois elas serão vistas com detalhes nos capítulos seguintes, onde analisamos um problema concreto e resolvê-lo-emos seguindo nosso fluxograma. O que importa agora é ficar claro que o esforço a ser feito para que todas as fases sejam executadas, aparentemente inútil, será bastante compensador no futuro e que, com certeza, o tempo total empregado para elaborar o programa é bem menor, se este for feito respeitando-se todas as fases.

## EXERCÍCIO

1. Como único exercício para treinar o raciocínio a ser utilizado, elabore um programa capaz de "ler" o nome de até vinte alunos de uma escola e as notas correspondentes a três matérias para quatro bimestres (ou seja, 12 notas por aluno). O programa deve colocar os alunos em ordem alfabética, calcular sua média por matéria, a média geral e indicar quais são os alunos que foram reprovados. Deve possibilitar também uma saída, onde os alunos são colocados em ordem decrescente de nota (média geral), para possibilitar a premiação dos três primeiros colocados.






Neste capítulo não usaremos o computador e pouco falaremos nele. Entretanto, este é o capítulo que deve ser lido com mais cuidado. Nele vamos expor um problema concreto e vamos resolvê-lo "manualmente".

Simulando o trabalho que o computador terá, poderemos tomar a consciência exata de todos os passos que a resolução do problema envolve. Vamos então à apresentação do problema:

Supondo que o leitor trabalhe com o computador numa firma que resolveu confeccionar "Cestas de Natal" para seus funcionários. As cestas são de vários tipos (vamos supor, poeticamente, que as mais fartas serão para os funcionários mais humildes) e contêm vários tipos de produtos. Por uma questão de simplicidade, vamos supor apenas três tipos de cestas e quatro produtos diferentes (figura 2.1). O programa a ser implementado, porém, deverá deixar à escolha do usuário a quantidade de tipos e os produtos contidos em cada uma.

| Tipo \ Produto | Vinho | Nozes | Chocolate | Panetone |
|---|---|---|---|---|
| X | 1 | 2 | 1 | 1 |
| Y | 2 | 3 | 0 | 3 |
| Z | 2 | 2 | 1 | 1 |

Figura 2.1 — Tabela cestas x produtos

A tabela da figura 2.1 tem a óbvia forma de uma matriz de três linhas e quatro colunas. Cada elemento desta matriz, que vamos chamar de A, será simbolizado por A (C,P), onde C é o número que indica o tipo de cesta e P é o número que indica o tipo de produto. O valor do elemento A(C,P) será então a quantidade do produto P que consta em cada cesta do tipo C. Atribuindo os numerais 1, 2 e 3 às cestas X, Y e Z e os numerais 1, 2, 3 e 4 aos produtos VINHO, NOZES, CHOCOLATE e PANETONE, respectivamente, a tabela da figura 2.1 passará a ter o aspecto da tabela da figura 2.2.

| C \ P | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 1 |
| 2 | 2 | 3 | 0 | 3 |
| 3 | 2 | 2 | 1 | 1 |

Figura 2.2 — Matriz A

Antes de continuarmos a exposição do problema, seria conveniente testarmos se a simbologia usada está sendo entendida.

Portanto, vamos propor um exercício cuja resposta está no fim do capítulo.

EXERCÍCIO 1: Preencha os espaços vazios.

a) A(2,4) = 3
b) A(3,3) = 1
c) A (2 , 3) = 0
d) A cesta tipo Y tem 3 panetones, logo:
A(2 , 4) = 3
e) O valor final de D será 0 .

Todo material necessário para a confecção de cada tipo de cesta deverá ser comprado de um único fornecedor. Para escolher o fornecedor mais conveniente, o leitor-comprador efetua uma pesquisa em cinco supermercados. Fixamos o número de fornecedores em cinco por uma questão de simplicidade, mas o programa a ser implementado deve deixar este número em



aberto à escolha do usuário. O resultado da pesquisa de preços unitários está resumido na tabela da figura 2.3.

| Produto \ Loja | Eldorado | Morita | Gonçalves Sé | Pão de Açúcar | Carrefour |
|---|---|---|---|---|---|
| Vinho | 1000 | 1100 | 980 | 1000 | 970 |
| Nozes | 800 | 850 | 900 | 750 | 950 |
| Chocolate | 900 | 1000 | 900 | 850 | 850 |
| Panetone | 2000 | 2200 | 1900 | 2000 | 2100 |

Figura 2.3 — Tabela produtos x lojas

Chamando a matriz correspondente de B, onde B(P,L) é o custo do produto P na loja L, obteremos a tabela da figura 2.4.

| P \ L | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1000 | 1100 | 980 | 1000 | 970 |
| 2 | 800 | 850 | 900 | 750 | 950 |
| 3 | 900 | 1000 | 900 | 850 | 850 |
| 4 | 2000 | 2200 | 1900 | 2000 | 2100 |

Figura 2.4 — Matriz B

EXERCÍCIO 2: Preencha os espaços vazios:

a) B(2,5) = 950
b) B (3 , 2) = 1000
c) No Morita, cada panetone custa 2200 , logo:
B(4 , 2) = 2200

Chegamos agora ao ponto crucial do programa. Para descobrir qual o fornecedor mais conveniente para cada tipo de cesta, devemos preencher a tabela da figura 2.5.

| Loja / Cesta | Eldorado | Morita | Gonçalves Sé | Pão de Açúcar | Carrefour |
|---|---|---|---|---|---|
| X | | | | | |
| Y | | | | 10250 | |
| Z | | | | | |

Figura 2.5 — Tabela cesta x lojas

Consultando-se a tabela na figura 2.1, observamos que a cesta tipo Y tem:

Y = 2 VINHOS + 3 NOZES + 0 CHOCOLATES + 3 PANETONES

O custo da cesta Y do Pão de Açúcar foi calculado da seguinte forma: tendo em vista que cada vinho custa 1000 (veja tabela na figura 2.3), cada pacote de nozes 750, cada chocolate 850 e cada panetone 2000, o custo da cesta Y será:

Y = 2 × 1000 + 3 × 750 + 0 × 850 + 3 × 2000 = 10250

Definimos agora uma matriz C correspondente a tabela da figura 2.5, de modo que cada um dos seus elementos C(C,L) represente o custo da cesta tipo C na loja número L. Olhando a tabela na figura 2.5, vemos então que C(2,4) = 10250.

EXERCÍCIO 3: Preencha a tabela da figura 2.6, anotando os cálculos efetuados numa folha à parte.

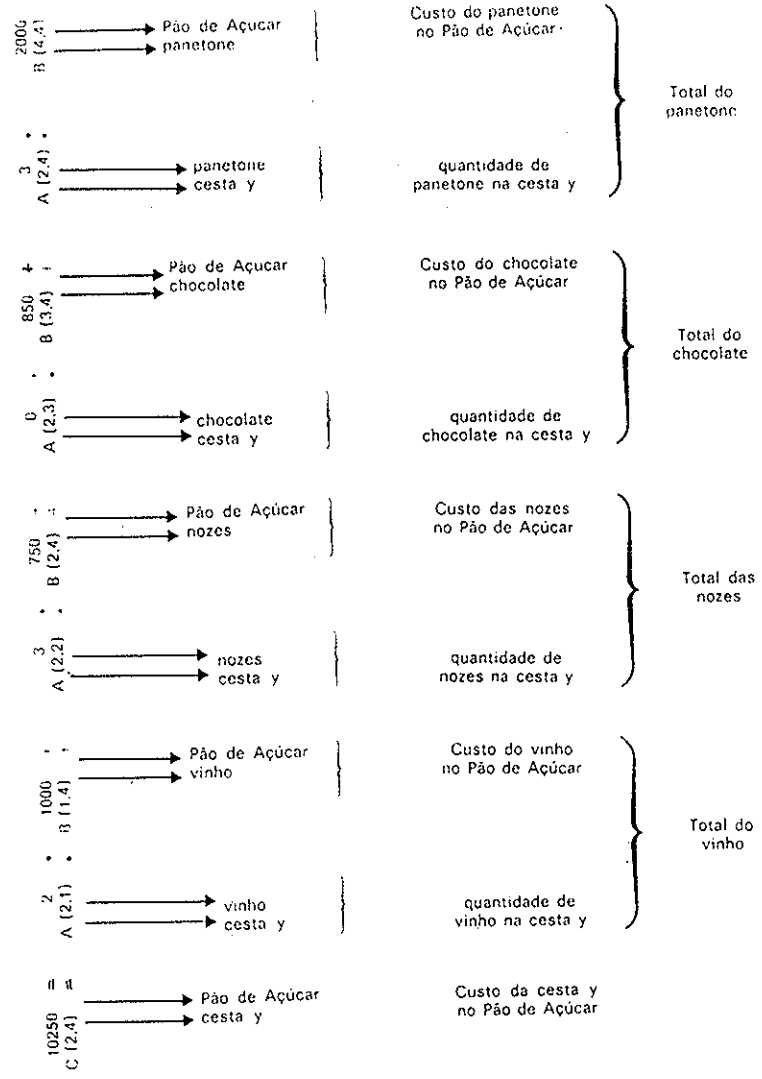
| C \ L | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 5500 | 6000 | 5500 | 3350 | 5820 |
| 2 | | | | 10250 | |
| 3 | | | | | |

Figura 2.6 — Matriz C



Vamos agora recalcular o preço da cesta tipo Y no Pão de Açúcar, usando a simbologia das matrizes:

Figura 2.7



O leitor-comprador deve ter notado que com essa simbologia podemos determinar quais são os elementos das matrizes A e B que devem ser operados para se obter um elemento da matriz C:

C(R,S) = A(R,1) × B(1,S) + ..... + A(R,NP) × B(NP,S)

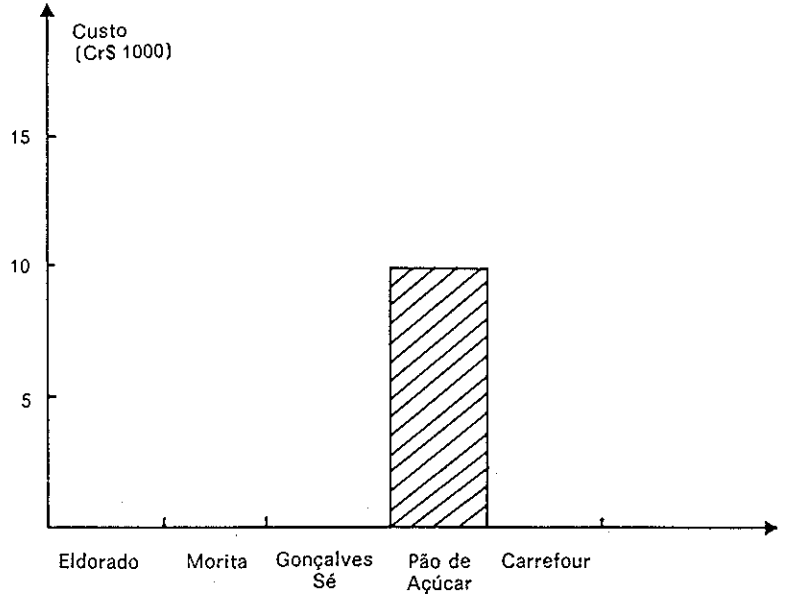
onde NP é o número de produtos diferentes.

EXERCÍCIO 4: Com base no exemplo dado (três tipos de cestas, quatro produtos e cinco supermercados), preencha as sentenças a seguir:

a) C(3,5) = A(3,1) × B(1,5) + A(3,2) × B(2,5) + .....
b) C(2,3) = A(..... , .....) × B(..... , .....) + .....
c) C(M,N) = A (..... , .....) × B(..... , .....) + .....

EXERCÍCIO 5: Responda as perguntas a seguir:

a) Qual o fornecedor mais conveniente para a cesta tipo X?
b) Qual o fornecedor menos conveniente para a cesta tipo Z?
c) Complete o histograma para a cesta Y.






Tendo resolvido o problema "à unha", o leitor deve agora ser capaz de entender melhor o desenvolvimento do programa correspondente que será implementado nos próximos capítulos.

## RESPOSTAS DOS EXERCÍCIOS

EXERCÍCIO 1:

a) A(2,4) = **3**
b) A(3,3) = **1**
c) A(**2,3**) = 0
d) A cesta Y tem 3 panetones, logo A(**2,4**) = **3.**

EXERCÍCIO 2:

a) B(2,5) = **950**
b) B(**3,2**) = 1000
c) No Morita cada panetone custa 2200, logo B(**4,2**) = 2.200.

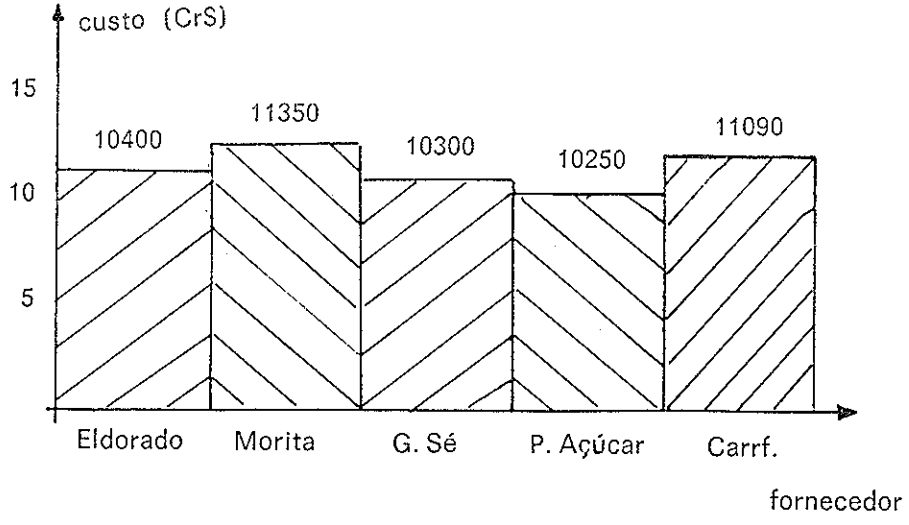
EXERCÍCIO 3:

| C \ L | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 5500 | 60000 | 5580 | 5350 | 5820 |
| 2 | 10400 | 11350 | 10300 | 10250 | 11090 |
| 3 | 6500 | 7100 | 6560 | 6350 | 6790 |

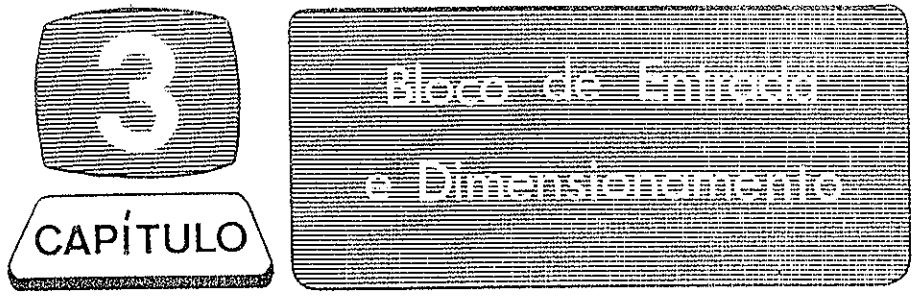
EXERCÍCIO 4:

a) C(3,5) = A(3,1).B(1,5) + A(3,2).B(2,5) + A(3,3).B(3,5) + + A(3,4).B(4,5)
b) C(2,3) = A(2,1).B(1,3) + A(2,2).B(2,3) + A(2,3).B(3,3) + + A(2,4).B(4,3)
c) C(M,N) = A(M,1).B(1,N) + A(M,2).B(2,N) + A(M,3). B(3,N) + A(M,4).B(4,N)

EXERCÍCIO 5:
a) fornecedor 4 — Pão de Açúcar
b) fornecedor 2 — Morita
c)






Este capítulo inicia uma série de quatro destinados a "resolver" passo a passo o problema descrito no capítulo anterior. Obviamente, iremos seguir os "conselhos" e o fluxograma apresentados no capítulo 1 e iniciaremos então, quebrando o problema em cinco blocos principais:

1. **Bloco de entrada e dimensionamento**

Responsável pela interface de entrada entre o programa e o usuário, ou seja, é a parte do programa que "pergunta" os dados a serem processados e coloca-os convenientemente na memória do computador. Nesta parte (e em qualquer outra parte que exija "respostas" do usuário), muito cuidado deve ser tomado para que o programa não "pare" devido à introdução de dados não adequados, ou seja, o programa deve ser "idiot proof".

2. **Bloco de cálculo do preço nas lojas**

Responsável pelo cálculo do produto matricial que irá gerar a matriz loja x cesta. Uma vez entendido o conceito matemático, este provavelmente é o bloco mais simples de ser programado, pois não há interação com o usuário.

3. **Bloco de cálculo do preço mínimo**

Responsável por determinar qual a loja mais conveniente para a aquisição das cestas. É conveniente que este bloco seja uma sub-rotina de modo a possibilitar seu uso em dois tipos de "saídas": tabela e gráfica.

4. **Bloco de saída-tabela**

Responsável por indicar na tela e/ou impressora o resultado do problema em forma de tabela, ressaltando-se a loja mais conveniente.

5. **Bloco de saída-gráficos**

Responsável por fornecer o mesmo tipo de informação do bloco anterior, porém na forma gráfica.

Cada um desses blocos será detalhado oportunamente. O que importa agora é ter uma idéia geral do desmembramento do problema e notar que cada bloco interage fortemente com o outro. A figura 3.1 apresenta um esquema dos blocos definidos. Poderíamos chamá-lo de fluxograma geral!

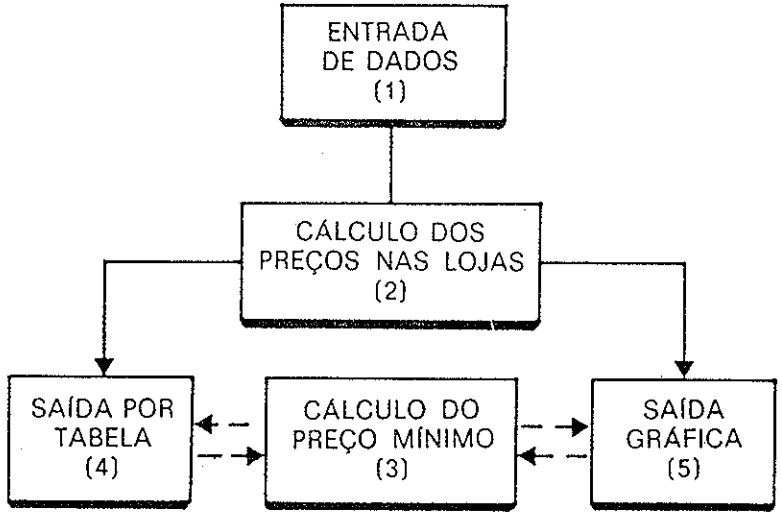



Figura 3.1 — Fluxograma geral do problema "Cestas de Natal"

Neste ponto podemos dizer que praticamente realizamos a fase 1 do desenvolvimento do software, ou seja, a definição do problema e o desmembramento em blocos principais. Resta ainda a descrição mais detalhada da função de cada bloco, mas isto será feito individualmente para cada um deles, assim como o



fluxograma, lista de variáveis, programação, teste e documentação. Somente ao terminar o último bloco é que será realizado o teste "geral" do programa.

Vamos então detalhar o bloco de entrada. As funções principais desses blocos são:

- ler e armazenar o número de cestas, de tipo, de produtos e de lojas;
- idem, para os nomes dos produtos e das lojas;
- idem, para a quantidade de cada produto por cesta e para o preço de cada produto por loja;
- verificar os dados introduzidos e permitir uma eventual modificação.

Como foi sugerido no capítulo interior, estes dados serão todos armazenados em forma de matrizes. Podemos então definir as seguintes variáveis:

**Principais:**

| | |
|---|---|
| NC | número de cestas |
| NP | número de tipos de produtos |
| NL | número de lojas |
| X(NC,NP) | matriz cestas x produtos |
| Y(NP,NL) | matriz produtos x lojas |
| P$(NP,10) | matriz com o nome dos produtos |
| L$(NL,10) | matriz com o nome das lojas |

**Secundárias (ou auxiliares)**

| | |
|---|---|
| I, J, K | contadores que podem ser modificados por outros blocos |
| A$ | variável para armazenar INKEY$ |
| IN | variável auxiliar para a sub-rotina de entrada numérica |
| INUM | indicadora do endereço da sub-rotina de entrada numérica |

As variáveis principais são as "variáveis-chaves" e normalmente serão utilizadas por outros blocos. Sendo assim, elas devem reaparecer nas outras listas com uma observação indicando "veja bloco 1" (onde, 1 = entrada de dados, veja figura 3.1).

As variáveis secundárias não são "transferidas" de um bloco para outro e podem inclusive ser modificadas por outros blocos. Aliás, é aconselhável que elas sejam "reutilizadas" por

outros blocos, pois cada nova variável definida, acarreta em ocupação de espaço de memória. No caso, I, J e K são apenas contadores, enquanto que as demais variáveis são utilizadas numa sub-rotina "idiot-proof", responsável por aceitar apenas dados numéricos durante um INPUT numérico. Desse modo, evita-se a "parada" do programa por erro se ao ser executado um INPUT numérico, o usuário inadvertidamente introduzir um dado alfanumérico (isto será visto em detalhes logo mais).

No caso de existirem diferentes blocos de entrada, uma sub-rotina poderia ser reutilizada por outros blocos e essas variáveis deixariam de ser auxiliares. Eventualmente, a própria sub-rotina poderia ser um bloco separado.

Quando o espaço na memória for crítico, deve-se inclusive procurar quais as variáveis comuns apenas a um determinado número de blocos (que não serão utilizados novamente) e permitir a sua reutilização por outros blocos. Felizmente, este não é o nosso caso. Note que quanto mais complexo e maior for o programa, surge então uma hierarquia de variáveis que deve ser muito bem especificada. Sugere-se inclusive a definição de algum tipo de código para os diferentes "níveis" de variáveis.

Voltemos então às "Cestas de Natal" e falemos sobre o dimensionamento das matrizes.

Nesse computador é possível o dimensionamento com variáveis (exemplo, DIM X(NC,NP)), mas isto não é aplicável a todos os computadores! Nesse caso, uma quantidade máxima para cada item deveria ser definida e isto deveria constar na documentação do programa (manual de instruções).

Nota-se que o tamanho do nome dos produtos e lojas foi restringido a dez caracteres. Isto deverá ser especificado nas "instruções" do programa (fase de documentação e acabamento). Além disso, devido a uma limitação desse computador, não podemos atribuir um nome mais conveniente às matrizes, pois elas podem ter apenas um caractere. De fato, além de fazer a lista de variáveis, é conveniente que elas sejam escolhidas (quando possível) de maneira a formar um mnemônico que ajude a lembrar sua função. Este procedimento facilita bastante a fase de teste e inclusive a análise futura do programa.

Elaboremos então o fluxograma. Como idéia inicial (e inclusive para testar o raciocínio adotado) vamos possibilitar a visualização na tela dos dados introduzidos após efetuadas todas as leituras, além de permitir eventuais alterações dos dados quando necessário. É interessante notar que a idéia de iniciar com



um esquema básico para teste com implementações "futuras" é seguido para cada bloco individualmente, quase como se fossem programas independentes.

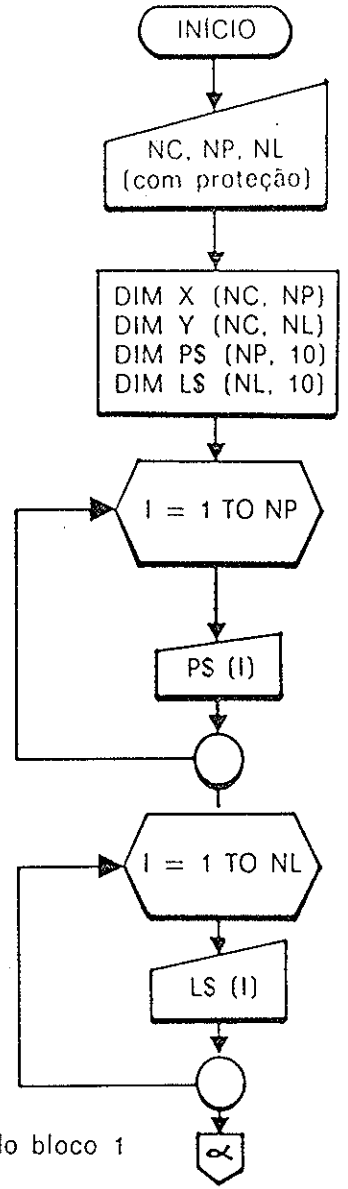
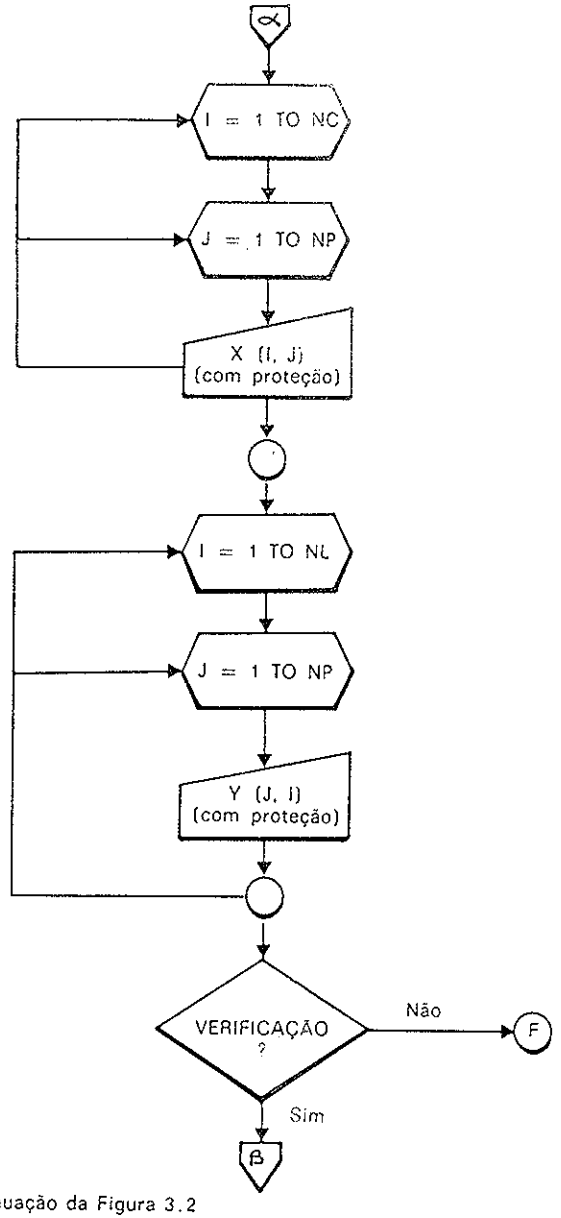
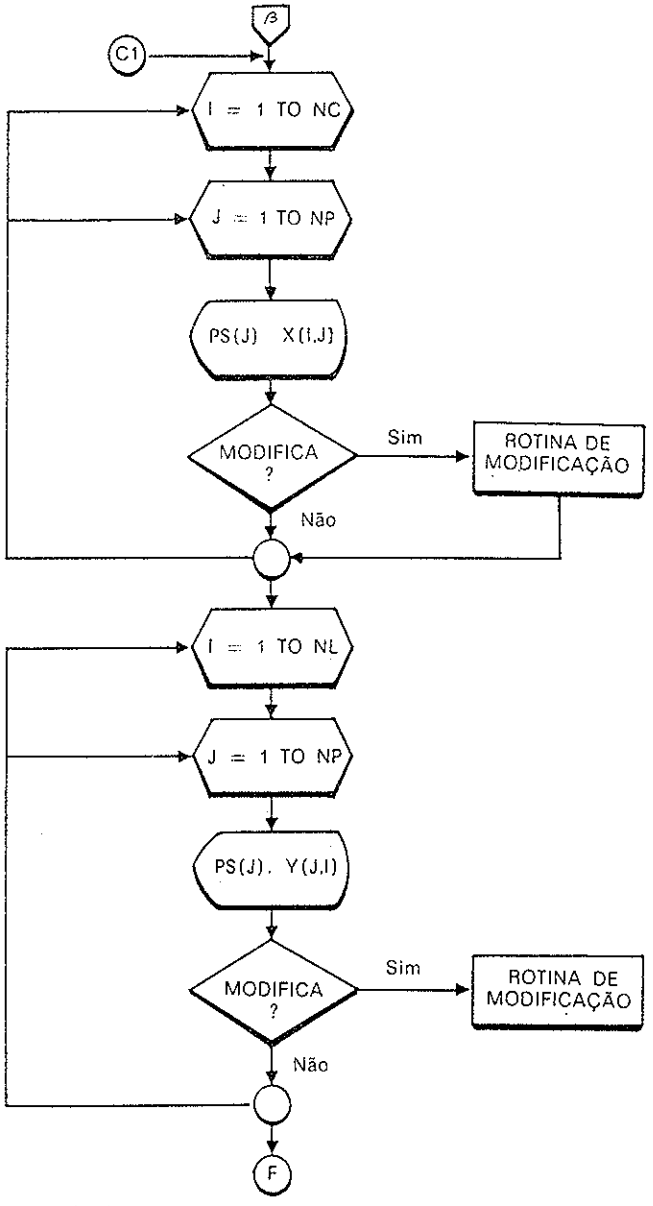



Figura 3.2
Fluxograma do bloco 1




Continuação da Figura 3.2






Continuação da Figura 3.2

Nota: o ponto de entrada C1 está relacionado com os blocos 4 e 5 e será explicado mais tarde (capítulo 5 e 6).

Procure seguir o fluxograma para entender o raciocínio adotado. Note que nenhum "detalhe" aparece no fluxograma para facilitar a visualização (de fato, um excesso de informações no fluxograma atrapalha mais do que ajuda). No entanto, a decisão do que é detalhe e do que é "importante" é bastante subjetiva e depende muito de cada um.

Antes de apresentar a listagem do programa (bloco), vamos analisar a sub-rotina utilizada em todos os blocos de leitura numérica (onde aparece escrito "com proteção"). Ela foi projetada para rejeitar qualquer caractere não numérico, ou seja, cujo código não esteja compreendido entre 28 e 37 (inclusive).

O valor numérico é colocado na variável IN.

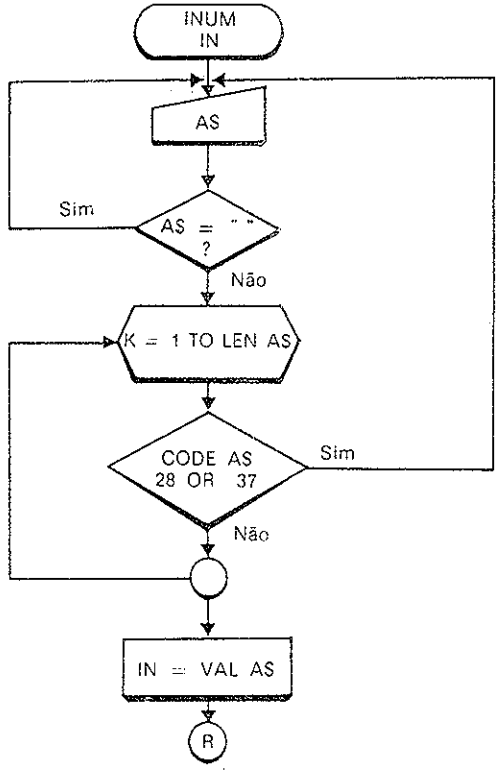



Figura 3 3 — Sub-rotina INUM ("idiot-proof")



Segue então a listagem do programa.

```
   0 REM ** PIAZZI - ROSSINI **
  10 REM CESTAS DE NATAL
  20 REM BLOCO DE ENTRADA DE DAD
OS
  30 SLOW
  40 LET INUM=1560
  50 PRINT "QUAL O NO.DE CESTAS
?"
  60 GOSUB INUM
  70 LET NC=IN
  80 PRINT NC
  90 PRINT
 100 PRINT "QUAL O NO.DE TIPOS D
E PRODUTOS"
 110 GOSUB INUM
 120 LET NP=IN
 130 PRINT NP
 140 PRINT
 150 PRINT "QUAL O NO.DE LOJAS C
ONSULTADAS ?"
 160 GOSUB INUM
 170 LET NL=IN
 180 PRINT NL
 190 PRINT
 200 PAUSE 600
 210 CLS
 220 DIM X(NC,NP)
 230 DIM Y(NP,NL)
 240 DIM P$(NP,10)
 250 DIM L$(NL,10)
 260 SCROLL
 270 PRINT "ENTRE COM OS NOMES D
OS PRODUTOS"
 280 SCROLL
 290 PRINT
 300 FOR I=1 TO NP
 310 SCROLL
 320 PRINT "PRODUTO ";I,
 330 INPUT P$(I)
 340 IF P$(I)="" THEN GOTO 330
 350 PRINT P$(I)
 360 NEXT I
 370 SCROLL
 380 PRINT
 390 SCROLL
 400 PRINT "ENTRE COM OS NOMES D
AS LOJAS"
 410 SCROLL
```

```
 420 PRINT
 430 FOR I=1 TO NL
 440 SCROLL
 450 PRINT "LOJA ";I,
 460 INPUT L$(I)
 470 IF L$(I)="" THEN GOTO 460
 480 PRINT L$(I)
 490 NEXT I
 500 SCROLL
 510 PRINT
 520 SCROLL
 530 PRINT "ENTRE COM A QUANTIDA
DE DE CADA"
 540 SCROLL
 550 PRINT "PRODUTO POR CESTA"
 560 FOR I=1 TO NC
 570 SCROLL
 580 PRINT
 590 SCROLL
 600 PRINT "CESTA ";I
 610 SCROLL
 620 PRINT
 630 FOR J=1 TO NP
 640 SCROLL
 650 PRINT "QUANT. DE ";P$(J);"?
";
 660 GOSUB INUM
 670 LET X(I,J)=IN
 680 PRINT IN
 690 NEXT J
 700 NEXT I
 710 SCROLL
 720 PRINT
 730 SCROLL
 740 PRINT "ENTRE COM O PRECO DE
 CADA "
 750 SCROLL
 760 PRINT "PRODUTO POR LOJA"
 770 FOR I=1 TO NL
 780 SCROLL
 790 PRINT
 800 SCROLL
 810 PRINT "LOJA ";L$(I)
 820 SCROLL
 830 PRINT
 840 FOR J=1 TO NP
 850 SCROLL
 860 PRINT P$(J);" ? CR$ ",
 870 GOSUB INUM
 880 LET Y(J,I)=IN
 890 PRINT IN
 900 NEXT J
 910 NEXT I
```



```
 920 SCROLL
 930 PRINT
 940 SCROLL
 950 PRINT "VERIFICACAO DOS DADO
S (S/N) ?"
 960 LET A$=INKEY$
 970 IF A$="" THEN GOTO 960
 980 IF A$="N" THEN GOTO 1530
 990 FAST
1000 CLS
1010 SLOW
1020 PRINT "VERIFICACAO   DIGITE
 "
1030 PRINT
1040 PRINT "S... P/SAIR"
1050 PRINT "C... P/CONTINUAR"
1060 PRINT "M... P/MODIFICAR"
1070 PAUSE 900
1080 CLS
1090 FOR I=1 TO NC
1100 SCROLL
1110 PRINT
1120 SCROLL
1130 PRINT "CESTA ";I
1140 SCROLL
1150 PRINT
1160 FOR J=1 TO NP
1170 SCROLL
1180 PRINT P$(J);"  ";X(I,J)
1190 LET A$=INKEY$
1200 IF A$="" THEN GOTO 1190
1210 IF A$="S" THEN GOTO 1530
1220 IF A$="C" THEN GOTO 1290
1230 IF A$<>"M" THEN GOTO 1190
1240 SCROLL
1250 PRINT "NOVO VALOR ?";
1260 GOSUB INUM
1270 LET X(I,J)=IN
1280 PRINT IN
1290 NEXT J
1300 NEXT I
1310 FOR I=1 TO NL
1320 SCROLL
1330 PRINT
1340 SCROLL
1350 PRINT "LOJA ";L$(I)
1360 SCROLL
1370 PRINT
1380 FOR J=1 TO NP
1390 SCROLL
1400 PRINT P$(J);" CR$ ";Y(J,I)
```

```
1410 LET A$=INKEY$
1420 IF A$="" THEN GOTO 1410
1430 IF A$="C" THEN GOTO 1510
1440 IF A$="S" THEN GOTO 1530
1450 IF A$<>"M" THEN GOTO 1410
1460 SCROLL
1470 PRINT "NOVO VALOR ? ";
1480 GOSUB INUM
1490 LET Y(J,I)=IN
1500 PRINT IN
1510 NEXT J
1520 NEXT I
1530 FAST
1540 CLS
1550 STOP
1560 INPUT A$
1570 IF A$="" THEN GOTO 1560
1580 FOR K=1 TO LEN A$
1590 IF CODE A$(K)<28 OR CODE A$
(K)>37 THEN GOTO 1560
1600 NEXT K
1610 LET IN=VAL A$
1620 RETURN
```

Vamos analisar rapidamente a listagem (um estudo mais profundo é bastante aconselhável ficando a cargo do leitor). Inicialmente, definimos INUM = 1560 por ser esse o endereço de entrada da sub-rotina INUM. Lemos então NC, NP e NL utilizando INUM. Repare cuidadosamente na utilização da variável IN, permitindo que a sub-rotina seja usada para qualquer entrada numérica.

Experimente executar o programa utilizando o exemplo do capítulo anterior. Tente introduzir algum dado "errado" para NC, NP ou NL (por exemplo, uma palavra ou um NEW LINE sem valor algum).

Após uma pequena pausa (colocada para possibilitar a percepção humana da instrução PRINT NL), limpamos a tela, dimensionamos as matrizes e iniciamos a leitura dos dados, sempre trabalhando com SCROLL para que não ocorra problemas de "fim de tela". Lemos então as matrizes PS, LS, X e Y. Com relação à matriz Y, um comentário é necessário, apesar dela ser "montada" como produtos x lojas. É conveniente que sua "leitura" seja feita, ao contrário, ou seja, lojas x produtos, pois normalmente temos a lista dos preços dos vários produtos por loja. Devido a isto, a introdução da linha 880 apresenta Y(J,I) em vez de Y (I,J).



Chegamos então à verificação dos dados, onde INKEY$ é utilizado para não ser necessário pressionar NEW LINE (tornando o programa mais elegante). Note que ao ser perguntado se a verificação é desejada ou não, apenas a tecla N causa a "saída" do programa. Todos os demais (S inclusive), causam o início da verificação (é importante usar o bom senso para definir com critério qual será a tecla "exclusiva", N no caso). De qualquer modo, isso poderia ser evitado e veremos isso logo mais.

Vamos supor então que a verificação foi solicitada. Limpamos então a tela, colocando o computador momentaneamente em FAST para evitar o indesejável efeito do CLS após a utilização do SCROLL (experimente retirar as linhas 990 e 1010). Apresentamos a seguir o "menu" dos comandos a serem utilizados durante a verificação:

S ..... para sair, ou seja, interromper a verificação e prosseguir para o próximo bloco (no caso "parar");

C ..... para continuar. Cada item será mostrado na tela e a tecla C fará com que o próximo item seja mostrado;

M ..... para modificar. Se uma modificação for necessária, a tecla M causará a entrada numa parte do programa que perguntará pelo novo valor e efetuará a modificação.

Mostramos então na tela os vários itens. Após cada item, o computador testa continuamente o INKEY$ para decidir qual ação deve tomar. Note que nessa estrutura, se qualquer outra tecla sem ser S, C ou M, for pressionada, o programa "fica onde está"!

Observação: a estrutura utilizada para o INKEY$, ou seja, a colocação numa string para, em seguida, fazer os testes nesta, é a ideal para esse tipo de aplicação, pois qualquer "problema" na teclagem (aperto duplo, teclas quase simultâneas, etc...) é eliminado. Ao ser solicitado M, é efetuada a modificação utilizando-se INUM novamente. Note a vantagem de utilizar-se uma variável para endereços de sub-rotinas. Além de facilitar a escrita do programa, permite que a sub-rotina seja escrita apenas no final do endereço mais conveniente. Durante o programa, foi necessário executar três vezes um GOTO para a saída, mas a priori não se conhece qual será a linha. Bastaria

então, ter colocado nas linhas 980, 1210 e 1440 um GOTO SAIDA, e no começo do programa, após ter "definido" o endereço de SAIDA, colocar LET SAIDA = 1530.

Este procedimento facilita tremendamente a escrita dos programas. Com isto terminamos o BLOCO DE ENTRADA. Uma releitura desse capítulo será necessária quando for feito o manual de instruções do programa.

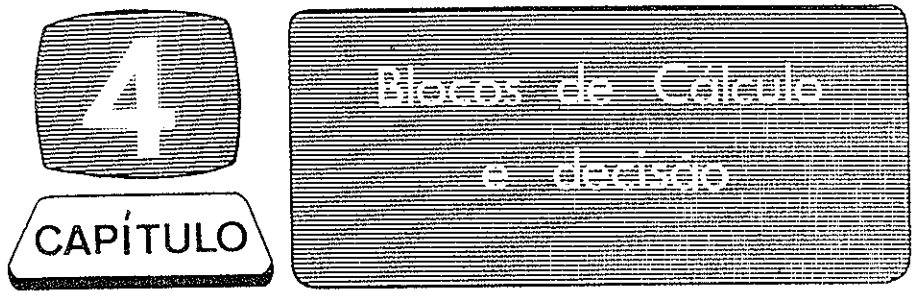
## EXERCÍCIOS

1) A sub-rotina INUM apresentada neste capítulo, além de rejeitar "letras", rejeita também números negativos ou com ponto decimal (ou vírgula). No nosso caso, isto não é um problema, pois não teremos preços ou quantidades negativas e, no caso dos preços, os centavos não valem mais nada... Entretanto, para outras aplicações este pode ser um aspecto negativo. Modifique então a sub-rotina para que ela aceite números negativos e decimais, inclusive se eles forem introduzidos com vírgula em vez de ponto.

2) O procedimento utilizado na parte de verificação (figura 3.1), para a modificação das matrizes X e Y é bastante semelhante. Transforme-o numa sub-rotina e modifique o final do programa para que ela seja utilizada.

3) Uma vez modificado algum valor, seria possível "testar" se o programa funcionou sem modificá-los. Em caso afirmativo, como?






Como já foi comentado anteriormente e supondo o raciocínio entendido, a parte do programa relativa aos cálculos é bastante simples de ser programada, conseqüentemente, esse capítulo será bem pequeno e acreditamos que também facilmente compreendido pelo leitor.

Apresentamos aqui os blocos 2 e 3, ou seja, o bloco de cálculo do preço nas lojas que envolve produto de matrizes e um loop triplo e o bloco de cálculo do preço mínimo.

Com relação ao bloco de cálculo do preço nas lojas, as suas funções são simplesmente:

- dimensionar uma matriz Z (NC,NL) que conterá o preço das cestas para cada loja;
- realizar o loop triplo para multiplicação das matrizes X e Y usando soma acumulativa na variável S, inicializada com zero;
- definir a variável MINIMAX com o endereço do bloco de cálculo do preço mínimo (mesmo truque utilizado com INUM), para tornar a listagem mais clara;
- perguntar ao usuário se ele deseja saída "normal" (tabelas) ou gráfica.

Nota: Essa última função é uma interação com o usuário, e no capítulo 3 afirmamos que essa interação não existia. De fato, essa função **não** faz parte do bloco em si, podendo ser considerada como um pequeno bloco de interface entre blocos que define um caminho a ser tomado. Aliás, dependendo do gosto do programador e da complexidade do programa, é conveniente fazer com que os blocos de interface sejam independentes dos blocos "principais".

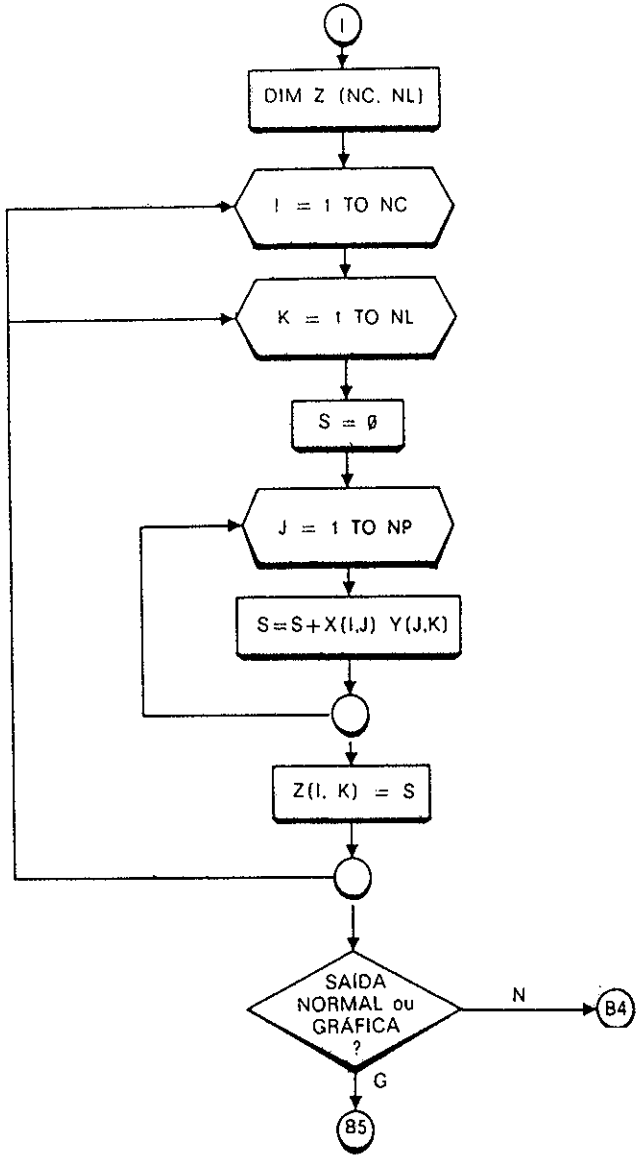



Figura 4.1 — Fluxograma do bloco 2



## VARIÁVEIS UTILIZADAS PELO BLOCO 2

### Principais

NC ..... número de cestas (veja bloco 1)
NL ..... número de lojas (veja bloco 1)
NP ..... número de tipos de produtos (vela bloco 1)
Z(NC,NL) ..... matriz preço das cestas x lojas
MINIMAX ..... endereço da rotina de cálculo de preço mínimo

### Secundárias (ou auxiliares)

I, J, K ..... contadores
S ..... variável para a soma acumulativa
A$ ..... variável para armazenar INKEY$

Nota: No bloco 1, a variável INUM poderia ser considerada principal se pensássemos em futuras expansões do programa que possam vir a gerar a necessidade de usar essa rotina (INUM) em outros blocos. Segue o fluxograma do bloco 2.

Figura 4.1

Note que em vez de usarmos o símbolo F para terminar o bloco, usamos os símbolos B4 e B5 para indicar que essa é a conexão com os blocos 4 e 5 respectivamente. Analogamente, no capítulo 3 poderíamos ter utilizado B2 em vez de F . Esse procedimento é bastante aconselhável, principalmente se o programa for muito grande. Aliás, os próprios símbolos I deveriam ser substituídos por B1 , B2 , B3 ,etc, para indicar os "pontos comuns". Se forem criados blocos de interface como foi sugerido anteriormente, estes poderiam, por exemplo, iniciar com os símbolos I1 , I2 , I3 , etc. e ter os símbolos dos blocos a acessar como saída (por exemplo, B4 e B5 nesse caso).

Vamos então à listagem do bloco de cálculo do preço nas lojas.

```
1700 REM ** BLOCO DE CALCULOS **
1710 PRINT AT 10,0;"AGUARDE - CA
LCULOS EM PROGRESSO"
1720 PAUSE 180
1730 DIM Z(NC,NL)
1740 FOR I=1 TO NC
```

```
1750 FOR K=1 TO NL
1760 LET S=0
1770 FOR J=1 TO NP
1780 LET S=S+X(I,J)*Y(J,K)
1790 NEXT J
1800 LET Z(I,K)=S
1810 NEXT K
1820 NEXT I
1830 LET MINIMAX=2000
1840 CLS
1850 SLOW
1860 PRINT "SAIDA NORMAL OU GRAF
ICA (N/G) ?"
1870 LET A$=INKEY$
1880 IF A$="N" THEN GOTO 2200
1890 IF A$="G" THEN GOTO 2800
1900 GOTO 1870
```

Acreditamos que não são necessárias explicações quanto à lógica adotada. Frisamos novamente o jeito simplificado de representar as linhas 1860, 1870, 1880, 1890 e 1900 no fluxograma.

Outro ponto a ser considerado é a possibilidade de incluir os comandos FAST e SLOW nos fluxogramas. Quanto aos detalhes na tela (por exemplo: CLS, SCROLL, ETC.) e à definição de variáveis para endereço de rotinas (INUM, MINIMAX, etc.) acreditamos que elas são perfeitamente dispensáveis.

Chegamos então ao bloco 3. Trata-se de uma sub-rotina para calcular qual a loja que tem o preço mínimo e qual a que tem o preço máximo. O preço máximo é necessário para definir corretamente as escalas durante a saída gráfica. Chamaremos essa rotina de MINIMAX.

## VARIÁVEIS UTILIZADAS PELO BLOCO 3

### Principais

MAX ..... valor do preço máximo
MIN ..... valor do preço mínimo
CX ..... número da loja correspondente a MAX
CN ..... número da loja correspondente a MIN
Z(NC,NL) ..... matriz preço das cestas x lojas (veja bloco 2)
C ..... número da cesta para a qual devem ser calculados MAX, MIN, CX e CN (veja blocos 4 e 5)



### Secundárias

I ..... contador

Esse bloco simplesmente utiliza o número da cesta fornecido pelos blocos 4 ou 5 em C e, fixada a "linha" da matriz Z, ele "pesquisa" todas as colunas para encontrar o valor máximo e o valor mínimo. Inicialmente supomos o primeiro elemento (primeira coluna), como sendo MINMAX e comparamos cada elemento a esse. Se o novo elemento for maior, MAX será esse novo elemento. Se o novo elemento for menor, MIN será esse novo elemento. No fim da "pesquisa" (loop) teremos MAX com o preço máximo e MIN com o preço mínimo. Analogamente, inicialmente faremos CX = CN = 1 e no final teremos CN indicando o número da loja com o preço mínimo e CX com o número da loja do preço máximo.

Esse raciocínio pode parecer meio confuso à primeira vista, mas acreditamos que uma boa análise do fluxograma e listagem a seguir esclarecerá as idéias. Nesse caso, incluímos os comandos FAST e SLOW no fluxograma.

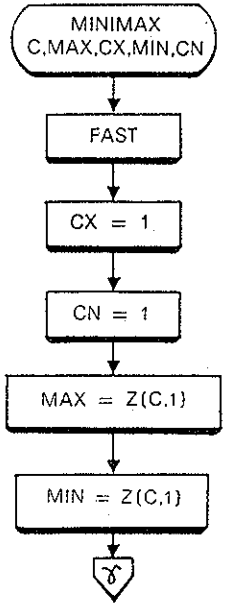
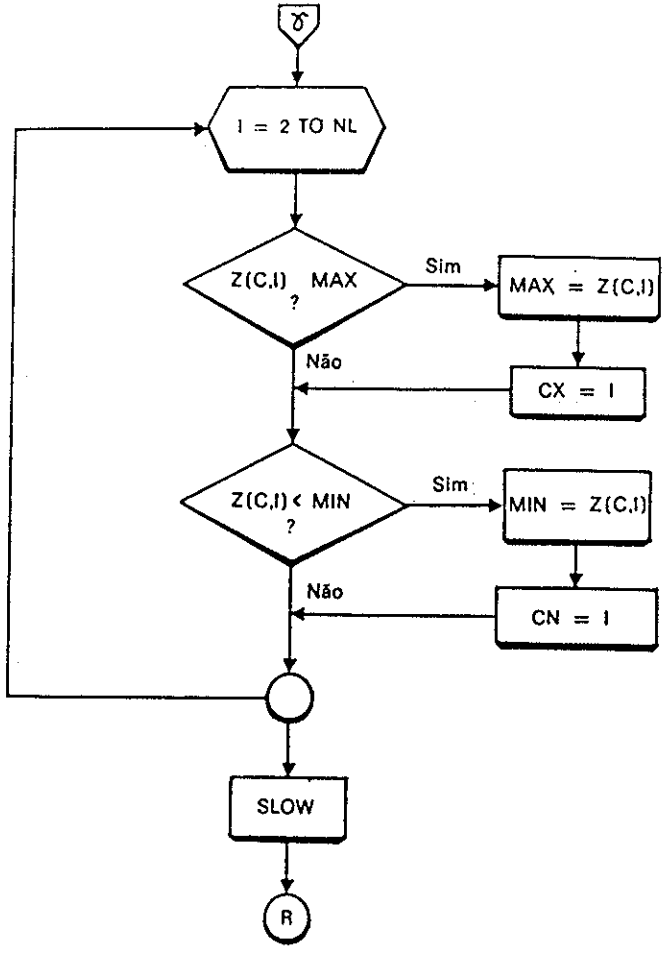



Figura 4.3 — Fluxograma do bloco 3




continuação da figura 4.3

Note que no símbolo de entrada da sub-rotina colocamos seu nome (MINIMAX) e os parâmetros de entrada (C) e saída (MAX, VX, MIN, CN). Poderá surgir a pergunta: a matriz Z(NC, NL) não deveria ser colocada? A resposta é **não**, pois não se trata de um parâmetro "diretamente" fornecido, mas de uma porção de memória de uso "comum" a diversos blocos e é por



isso que ela é listada como variável principal do bloco 3. Esse tipo de raciocínio é muito útil, caso você deseje estudar outras linguagens de programação.

```
2000 REM ** MINIMAX **
2010 FAST
2020 LET CX=1
2030 LET CN=1
2040 LET MAX=Z(C,1)
2050 LET MIN=Z(C,1)
2060 FOR I=2 TO NL
2070 IF Z(C,I)<=MAX THEN GOTO 21
00
2080 LET MAX=Z(C,I)
2090 LET CX=I
2100 IF Z(C,I)>=MIN THEN GOTO 21
30
2110 LET MIN=Z(C,I)
2120 LET CN=I
2130 NEXT I
2140 SLOW
2150 RETURN
```

Façamos um pequeno teste. Primeiramente para "conectar" os blocos, modifique a linha 1550 do programa na figura 3.4 no capítulo 3, por:

```
1550 GOTO 1700
```

Rode o programa, interrompa o loop final 1870-1900 e experimente imprimir algum elemento da matriz Z.

## EXERCÍCIOS

1) Considere a possibilidade de não utilizar MINIMAX como sendo chamada pelos blocos 4 e 5. Faça um programa que construa uma nova matriz que contém os preços mínimos e máximos e lojas correspondentes.

2) Existe alguma possibilidade de indicar os preços mínimos e máximos na própria matriz Z sem ter que construir outra matriz como sugerido no exercício 1? Como?

3) Os preços mínimos e máximos poderiam ser calculados "dentro" do loop triplo do bloco 2? Como?




Uma vez "construída" a matriz Z que contém os preços das cestas em todas as lojas, resta agora apresentar os resultados ao usuário. Nesse capítulo, iremos analisar o bloco que apresenta os resultados em forma de tabelas, ressaltando a loja na qual o preço da cesta é mínimo. Chamaremos esse bloco de "bloco de saída normal" (bloco 4).

O acesso a esse bloco é feito no final do bloco 2 (que poderia ser inclusive um bloco de interface independente) com o usuário pressionando a tecla N (Normal) durante o loop 1870-1900.

As principais funções desse bloco são:

- definir a variável CMIN com o endereço da rotina responsável por ressaltar a loja com o preço mínimo;
- perguntar ao usuário o número da cesta desejada (C);
- usar MINIMAX para "calcular" o preço mínimo e identificar a loja;
- imprimir na tela o preço da cesta desejada nas várias lojas;
- usar a rotina CMIN para ressaltar o preço mínimo;
- possibilitar ao usuário quatro tipos de "saídas" através de um loop de INKEY$:

| | |
|---|---|
| C | verificar outra cesta (não é saída) |
| G | ir ao bloco de saída-gráfica (bloco 5) |
| T | terminar o programa |
| V | retornar à verificação inicial (bloco 1) para permitir eventual modificação dos parâmetros |

Nota: como já sugerido no capítulo anterior, essa última função poderia ser um bloco de interface independente.

## VARIÁVEIS UTILIZADAS PELO BLOCO 4

### Principais

Z(NC,NL) ..... matriz preço das cestas x lojas (veja bloco 2)
NC ..... número de cestas (veja bloco 1)
CMIN ..... endereço da rotina que ressalta o preço mínimo
INUM ..... endereço da rotina "idiot-proof" (veja bloco 1)
C ..... número da cesta desejada (veja bloco 3)
MINIMAX ..... endereço da rotina de cálculo dos preços máximo e mínimo (veja bloco 3)
NL ..... número de lojas (veja bloco 1)
L$(NL,10) ..... matriz com o nome das lojas (veja bloco 1)
CN ..... número da loja com o preço mínimo (veja bloco 3)
CX ..... número da loja com o preço máximo (veja bloco 3)
H$(10) ..... string que contém o nome da loja de preço mínimo em vídeo-inverso

### Secundárias (ou auxiliares)

I, J ..... contadores
A$ ..... variável para armazenar INKEY$
IN ..... usada por INUM em entradas numéricas

Nota: CMIN e H$(10) foram consideradas variáveis principais supondo futura expansão do programa que use CMIN em outros blocos. Vamos então ao fluxograma desse bloco na figura 5.1.

Repare que no fluxograma utilizamos a notação sugerida no capítulo anterior para as entradas e saídas ( B4 , B5 , C1 ). C1 é o ponto de conexão com o bloco 1 (cap. 3, fig. 3.2) no caso em que o usuário decida optar por uma verificação e/ou modificação dos dados de entrada após ter analisado a saída.

Note que estamos ressaltando também (embora menos) a loja que tem o preço máximo. Não incluímos os blocos relativos a FAST ou SLOW, embora não houvesse problema em colocá-los.



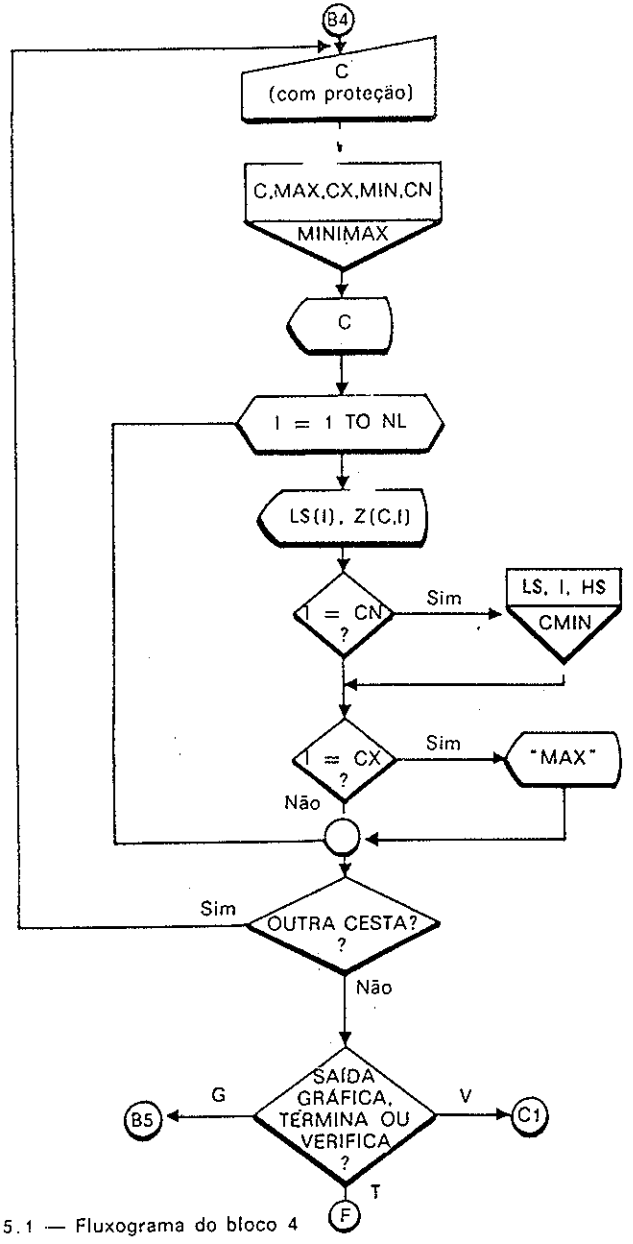
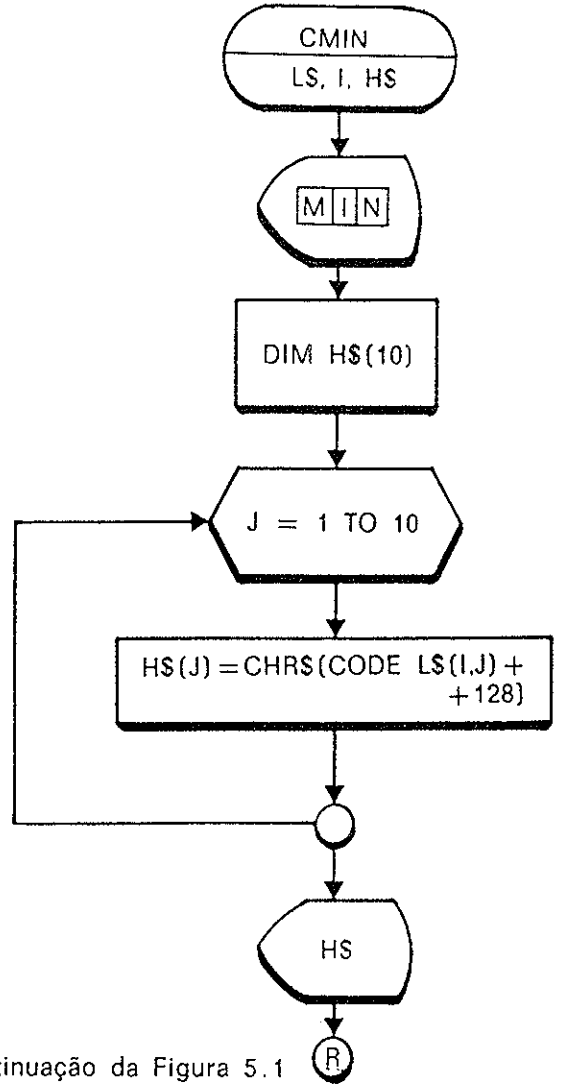



Figura 5.1 — Fluxograma do bloco 4




Continuação da Figura 5.1

Observe agora o fluxograma da sub-rotina CMIN. O bloco dentro do loop pode causar certa confusão na análise geral do programa que é a principal função do fluxograma. Ele poderia ser substituído por H$(J) = VIDEO-INVERSO (L$(I,J)).

Segue então a listagem do bloco 4:



```
2200 REM ** BLOCO DE SAIDA NORMA
L **
2210 FAST
2220 CLS
2230 SLOW
2240 LET CMIN=2650
2250 PRINT "NUMERO DA CESTA DESE
JADA ?"
2260 PRINT "(1 - ";NC;" ) ";
2270 GOSUB INUM
2275 LET C=IN
2280 PRINT C
2300 PAUSE 600
2310 GOSUB MINIMAX
2320 SCROLL
2330 PRINT "CESTA ";C
2340 SCROLL
2350 PRINT
2360 FOR I=1 TO NL
2370 SCROLL
2380 PRINT L$(I);"   CR$ ";Z(C,I
)
2390 IF I=CN THEN GOSUB CMIN
2400 IF I=CX THEN PRINT AT 21,28
;"MAX"
2410 NEXT I
2420 PAUSE 600
2430 FAST
2440 CLS
2450 SLOW
2460 PRINT "C... OUTRA CESTA"
2470 PRINT "G... SAIDA GRAFICA"
2480 PRINT "T... TERMINAR O PROG
RAMA"
2490 PRINT "V... VERIFICACAO INI
CIAL"
2510 PRINT
2520 PRINT "CESTA ";C
2530 PRINT
2540 PRINT L$(CX);"   CR$ ";Z(C,
CX);TAB 28;"MAX"
2550 PRINT
2560 PRINT AT 10,0,H$;"   CR$ ";
Z(C,CN);TAB 28;"MIN"
2570 LET A$=INKEY$
2580 IF A$="G" THEN GOTO 2800
2590 IF A$="V" THEN GOTO 990
2600 IF A$="C" THEN GOTO 2200
2610 IF A$="T" THEN GOTO 3150
2620 PRINT AT 10,0;L$(CN);"   CR
$ ";Z(C,CN);TAB 28;"MIN"
2630 GOTO 2560
```

```
2640 REM ** CMIN **
2650 PRINT AT 21,28;"[illegible]"
2660 DIM H$(10)
2670 FOR J=1 TO 10
2680 LET H$(J)=CHR$ (CODE L$(I,J
)+128)
2690 NEXT J
2700 PRINT AT 21,0;H$
2710 RETURN
```

Estude cuidadosamente a listagem e repare nas "simplificações feitas no fluxograma. Nesse ponto, podemos verificar se o programa funciona... Uma vez colocadas no computador as instruções correspondentes aos blocos 1, 2, 3 e 4, execute o programa usando, por exemplo, os parâmetros sugeridos no capítulo 2. Após o bloco de cálculo, selecione saída normal e verifique se os resultados correspondem aos esperados... Experimente obter a saída para as várias cestas e inclusive modificar os parâmetros iniciais.

Se você solicitar saída-gráfica (G) ou fim do programa (T), o programa pára. Para reiniciá-lo sem ter que entrar com todos os parâmetros novamente, **NÃO** digite RUN! Por exemplo, digite GOTO 990. Explique o por quê!

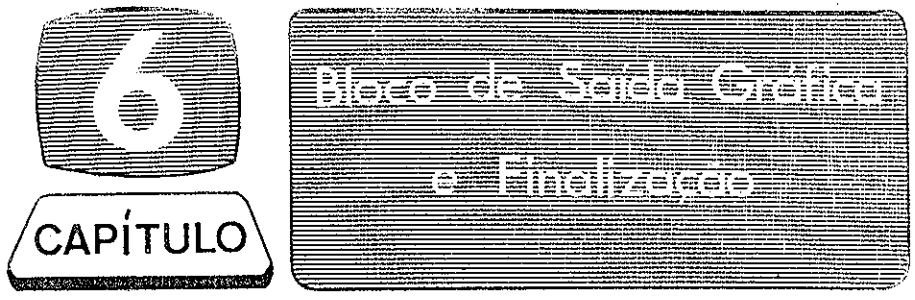
## EXERCÍCIOS

1) A sub-rotina CMIN exige uma pequena modificação para que possa ser usada "sem perigo" de um modo geral, ou seja, por outros blocos. Qual é essa modificação?

2) Você deve ter notado, ao trabalhar com a rotina de modificação, que ela só permite a alteração dos valores de produtos introduzidos no início. Complemente essa rotina para que ela permita a introdução de novos produtos e a retirada de produtos que não são mais desejados. Essa modificação irá trazer mudanças significativas ao bloco 1. Agora para fazer o DIM das matrizes X e Y deverá ser entrado o número máximo de itens desejados e outras variáveis deverão ser introduzidas para o número de itens introduzidos (que será menor ou igual ao máximo).

3) Estude alguma maneira de tornar mais rápido o loop 2560-2630.






Vamos agora finalizar o programa "Cestas de Natal", apresentando o bloco de saída-gráfica (bloco 5), além dos "retoques" finais do programa que, para maior rendimento do estudo, serão deixados a cargo do leitor.

Analogamente ao bloco 4, o acesso ao bloco 5 é feito no final do bloco 2 ou no final do bloco 4 através da tecla G. Obviamente, o bloco 5 deve oferecer possibilidade de acesso ao bloco 4 no final do mesmo.

Como principais funções temos:

- perguntar ao usuário o número desejado de cestas (C);
- usar MINIMAX para calcular os preços mínimo e máximo e respectivas lojas;
- definir uma "unidade gráfica" (S) baseado no valor do preço máximo;
- desenhar o gráfico em forma de barras para os preços da cesta, ressaltando o preço mínimo;
- possibilitar ao usuário quatro tipos de saídas através de um loop de INKEYS:

C ..... verificar outra cesta
N ..... ir para o bloco de saída normal
T ..... terminar o programa
V ..... retornar a verificação inicial

## VARIÁVEIS UTILIZADAS PELO BLOCO 5

### Principais

NC ..... número de cestas (veja bloco 1)

INUM ..... endereço da rotina "idiot-proof" (veja bloco 1)
C ..... número da cesta desejada (veja bloco 3)
MINIMAX ..... endereço da rotina de cálculo dos preços máximo e mínimo (veja bloco 3)
MAX ..... preço máximo
CN ..... número da loja com preço mínimo (veja bloco 3)
NL ..... número de lojas (veja bloco 1)
L$(10) ..... matriz com o nome das lojas (veja bloco 1)
Z(NC,NL) ..... matriz preço das cestas x lojas (veja bloco 2)

**Secundárias (ou auxiliares)**

IN ..... usada por INUM em entradas numéricas
S ..... indica a "unidade" gráfica
I, J ..... contadores
W ..... usada para localizar a posição (coluna) das barras
H ..... altura da barra em "unidades gráficas"
V$ ..... string usada para representar a unidade gráfica das barras (espaço em vídeo-inverso ou cifrão em vídeo-inverso para o preço mínimo)
A$ ..... variável para armazenar INKEY$

Iremos agora apresentar o fluxograma do bloco 5. Apesar de "pequeno", ele é razoavelmente complexo, pois utiliza vários "truques" para desenhar o gráfico desejado. Para definir a escala do programa, definimos a variável S. Como sabemos, a tela do computador é "dividida" normalmente em 22 linhas e 32 colunas, mas se fizermos POKE 16418,0, teremos à disposição 24 linhas e 32 colunas (veja vol. 1, cap. 11). Assim, se reservarmos a última linha para colocar o número das lojas, uma linha para colocar a altura da barra e quatro linhas no início para informações gerais (escala, número da cesta, nome da loja de preço mínimo), sobram 18 linhas para as barras. Teremos então S = MAX/18.

As barras devem ser desenhadas para cada loja e para



espaçá-las de duas colunas (por que?), usamos W = 3 * I — 3 como posição da barra, onde I é o número da loja. A altura de cada barra é dada por H = INT (Z(C,I)/S) e elas são construídas utilizando espaços em vídeo-inverso, exceto para o preço mínimo, onde cifrões em vídeo inverso são utilizados. O valor de H é colocado no final de cada barra.

O final é similar ao bloco 4 oferecendo ao usuário os quatro tipos de saída mencionados anteriormente e uma pequena animação para indicar que o computador (ou melhor, o programa) está funcionando. Aliás, esta é uma prática bastante aconselhável, ou seja, colocar algum tipo de animação, quando o computador está aparentemente parado durante o loop.

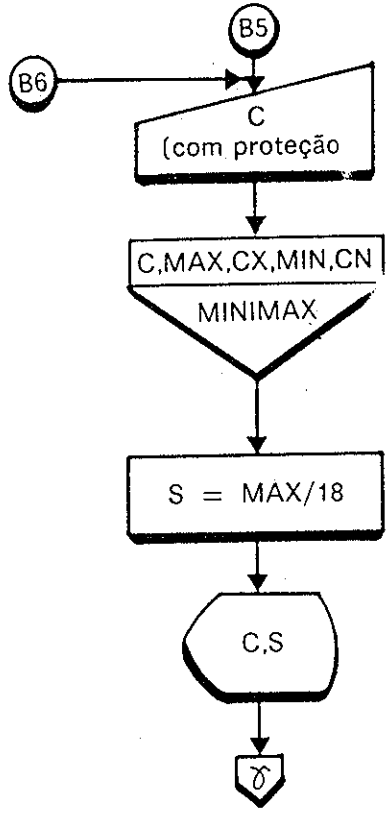
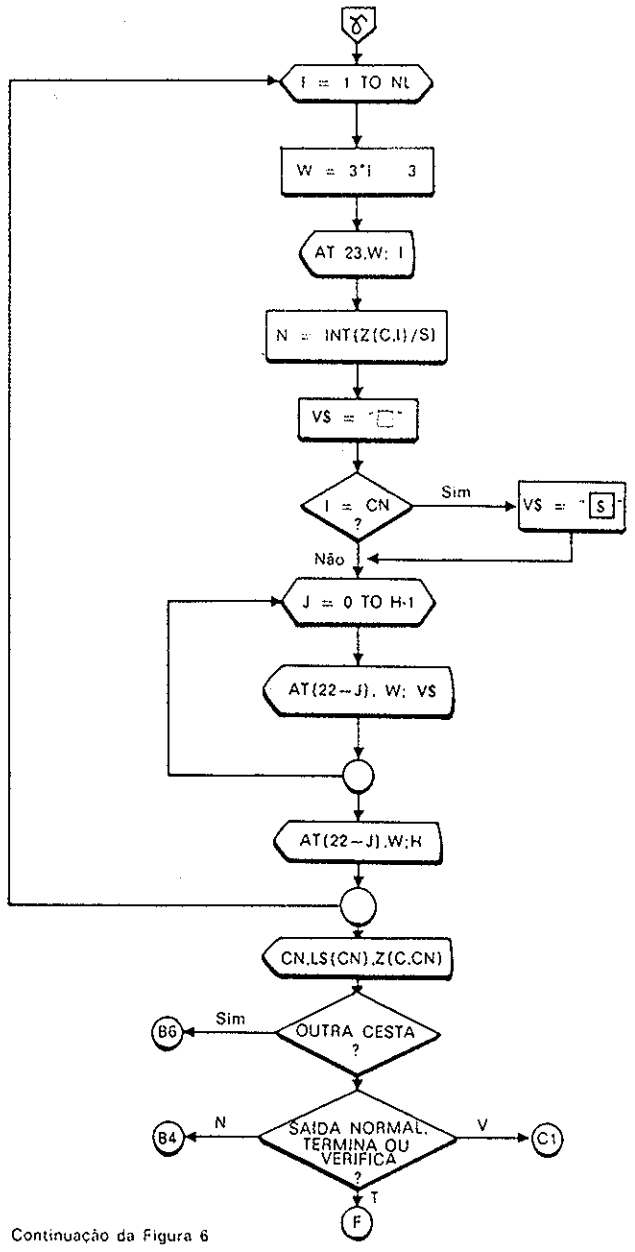



Figura 6.1 — Fluxograma do bloco 5




Continuação da Figura 6



Segue a listagem do bloco 5:

```
2800 REM ** BLOCO DE SAIDA GRAFI
CA **
2810 FAST
2820 CLS
2830 SLOW
2840 PRINT 'NUMERO DA CESTA DESE
JADA ?"
2850 PRINT "(1 - ";NC;" ) ";
2860 GOSUB INUM
2870 LET C=IN
2880 PRINT C
2890 PAUSE 600
2900 GOSUB MINIMAX
2910 LET S=MAX/18
2920 CLS
2930 PRINT " CESTA ";C,TAB 12;"
 = CR$ ";INT S;"  C/N/T/V"
2940 POKE 16418,0
2950 FOR I=1 TO NL
2960 LET W=3*I-3
2970 PRINT AT 23,W;I
2980 LET H=INT (Z(C,I)/S)
2990 LET V$="
3000 IF I=CN THEN LET V$="
3010 FOR J=0 TO H-1
3020 PRINT AT (22-J),W;V$
3030 NEXT J
3040 PRINT AT (22-J),W;H
3050 NEXT I
3060 POKE 16418,2
3070 PRINT AT 2,0;"
";CN;" ";
L$(CN);" CR$ ";Z(C,CN);"
$"
3080 LET A$=INKEY$
3090 IF A$="N" THEN GOTO 2200
3100 IF A$="V" THEN GOTO 990
3110 IF A$="C" THEN GOTO 2800
3120 IF A$="T" THEN GOTO 3150
3130 PRINT AT 2,0;"MIN ";CN;" ";
L$(CN);"
";Z(C,CN);"
$"
3140 GOTO 3070
3150 FAST
3160 CLS
3170 PRINT AT 12,5;"FIM DO PROGR
AMA"
3180 STOP
```

Novamente insistimos em que o leitor estude com detalhes a listagem e a relacione ao fluxograma apresentado. Note que após ter desenhado o gráfico, nós corrigimos o valor da memória 16418 fazendo POKE 16418, 2.

Nesse ponto o programa está completo e sugerimos que você execute-o, testando todas as possibilidades. Após pressionar a tecla T para terminá-lo, se você não deseja reiniciar todos os parâmetros, lembre-se de comandar GOTO para tanto.

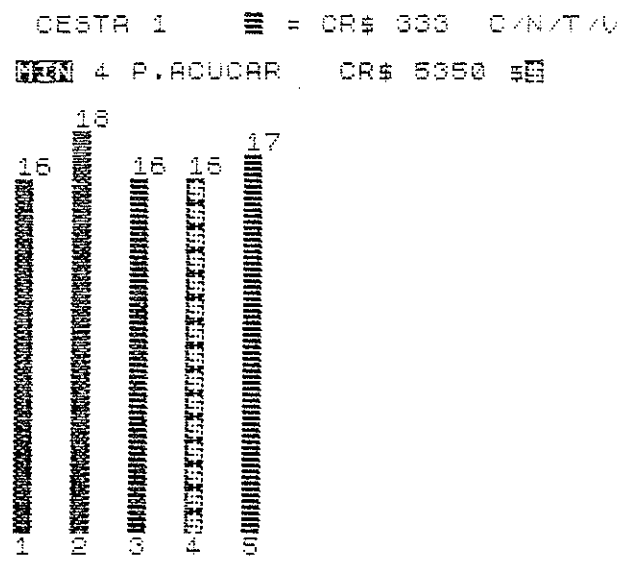



Figura 6.3 — Exemplo de saída gráfica do programa "Cestas de Natal" — note os códigos C/N/T/V correspondentes às saídas no canto superior direito.

Bom, como dissemos o programa está completo... Isso no ponto de vista do livro! Existem vários detalhes que podem ser implementados para tornar o programa mais "profissional". Além disso, dê uma olhada no capítulo 1. Achamos que há muito a ser feito quanto à colocação de comentários nas listagens (e fluxograma inclusive) e à documentação. De fato, um manual



deve ser escrito e isso fica a cargo do leitor, mas só depois de ter entendido com detalhes o programa apresentado e depois de tê-lo implementado suficientemente. Que implementações fazer? Comece pelo capítulo 3 e resolva **todos** os exercícios propostos até agora, a partir daí teremos uma série de exercícios nesse capítulo que darão os "retoques" finais... Naturalmente, a imaginação é sempre bem vinda e qualquer coisa que você deseje adicionar para melhorar o programa será ótimo. Quem sabe você não acabe até elaborando um programa suficientemente bom para explorá-lo comercialmente? Não se esqueça, porém, de mencionar esse livro, O.K.?

## EXERCÍCIOS

1) Resolva **todos** os exercícios dos capítulos anteriores (1 a 5).

2) Adicione um bloco ao programa que coloque os produtos em ordem e as lojas em ordem alfabética.

3) Adicione em blocos que permitam ordenar as lojas em ordem crescente e/ou decrescente de preço.

4) Refaça o programa com as seguintes características:

a) dois níveis de fluxograma, o primeiro onde praticamente **não** apareçam comandos BASIC (veja, por exemplo, o cálculo de H$(10) na figura 5.1) e o segundo bem "próximo" ao BASIC, incluindo FAST, SLOW, CLS e posicionamento na tela.

b) torne as rotinas INUM, CMIN e a verificação em blocos independentes;

c) use blocos de interfaceamento;

d) ao fazer a lista das variáveis, indique todos os blocos em que elas aparecem entre parênteses. No final, faça uma lista geral das variáveis.

5) Qual o número máximo de lojas que pode ser utilizado para a saída gráfica? Existe a possibilidade de aumentar esse número? Como?

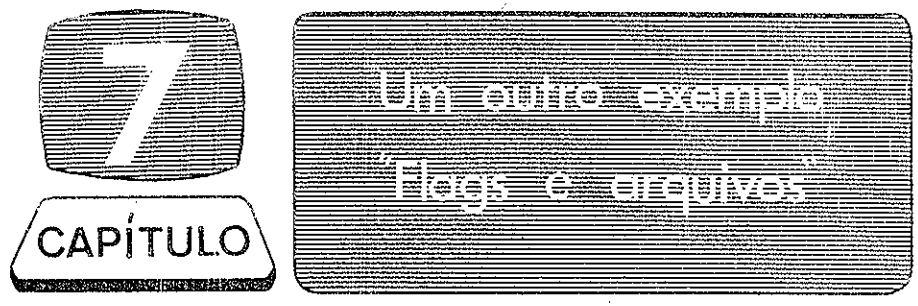
6) Adicione a possibilidade de um comando que gere um SAVE do programa (veja vol. 1, cap. 10). Ao fazer isso, as variáveis são armazenadas na fita juntamente com o programa e, ao fazer o LOAD do programa, as variáveis "vêm junto". Um simples GOTO após o comando SAVE poderia dirigir a um ponto do "novo programa", onde há uma opção entre re-utilizar as variáveis ou reiniciá-las.

7) Faça um novo bloco que seja capaz de localizar a loja mais conveniente para todas as cestas.

8) É possível "acelerar" o loop 3070-3140? Como?

9) Finalmente, escreva o manual do programa... boa sorte!






Em computação, é chamada de flag, uma variável que possui apenas duas condições: verdadeira ou falsa, usualmente simbolizadas por 1 ou 0.

As flags são utilizadas para indicar um resultado realizado por um determinado bloco dentro de um programa. A seguir a flag pode ser testada e com base no resultado obtido, pode ser tomada uma decisão.

Num programa extenso, com muitos desvios, uma flag no início de cada desvio torna possível, no final do programa, um controle do caminho seguido. Isto é muito útil para detectar erros de lógica.

Como exemplo, vamos elaborar um programa que ordena uma lista de itens com o respectivo preço e quantidade em ordem alfabética. Para facilitar a compreensão, o programa será introduzido em partes. Assim sendo, nos preocupamos inicialmente em ordenar alfabeticamente uma lista de nomes. Devido às limitações de memória e de tamanho de tela, o número máximo de letras por palavra será limitado (em computadores mais complexos esta limitação é facilmente contornável).

A lógica utilizada testa as palavras duas a duas. Se elas não estiverem corretamente posicionadas, elas têm suas posições trocadas e a operação é indicada numa flag. Ao fim da lista, a flag é testada. Se houver alguma troca a operação é repetida até que a flag indique que não houve trocas. Neste momento, todos os itens da lista estão corretamente posicionados.

Note que para trocar itens, é necessária uma variável auxiliar para guardar temporariamente um deles.

A seguir é apresentado o diagrama de blocos e a listagem da primeira parte do programa.

Observação: As palavras terão no máximo 14 letras e N = número máximo de itens (no caso igual a 25).

**Identificação da Função**

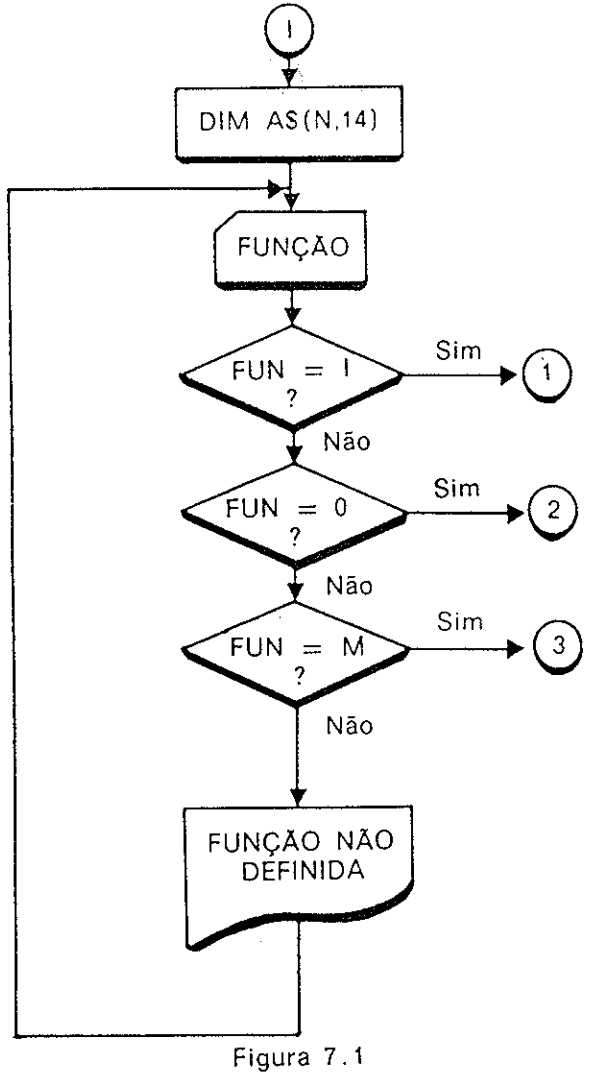
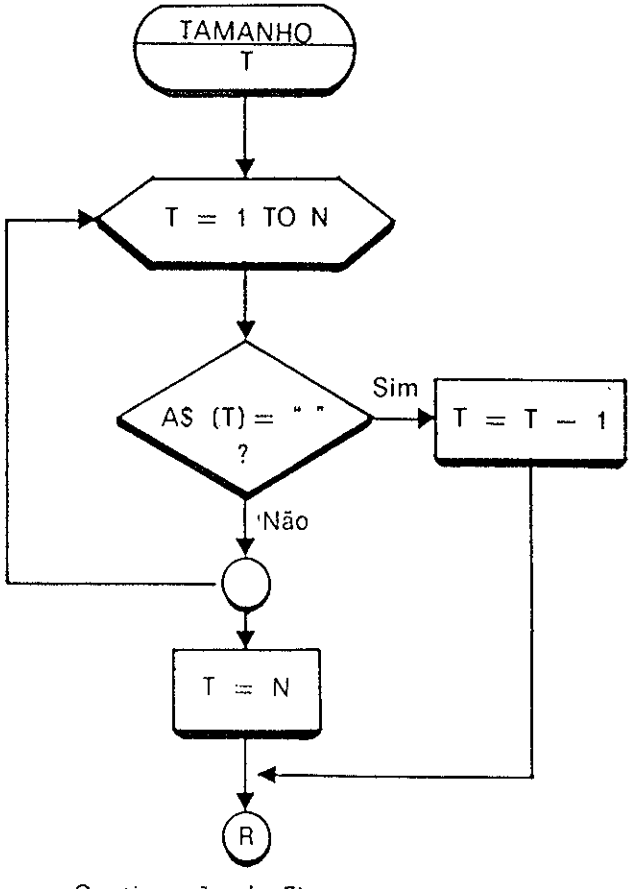



Figura 7.1



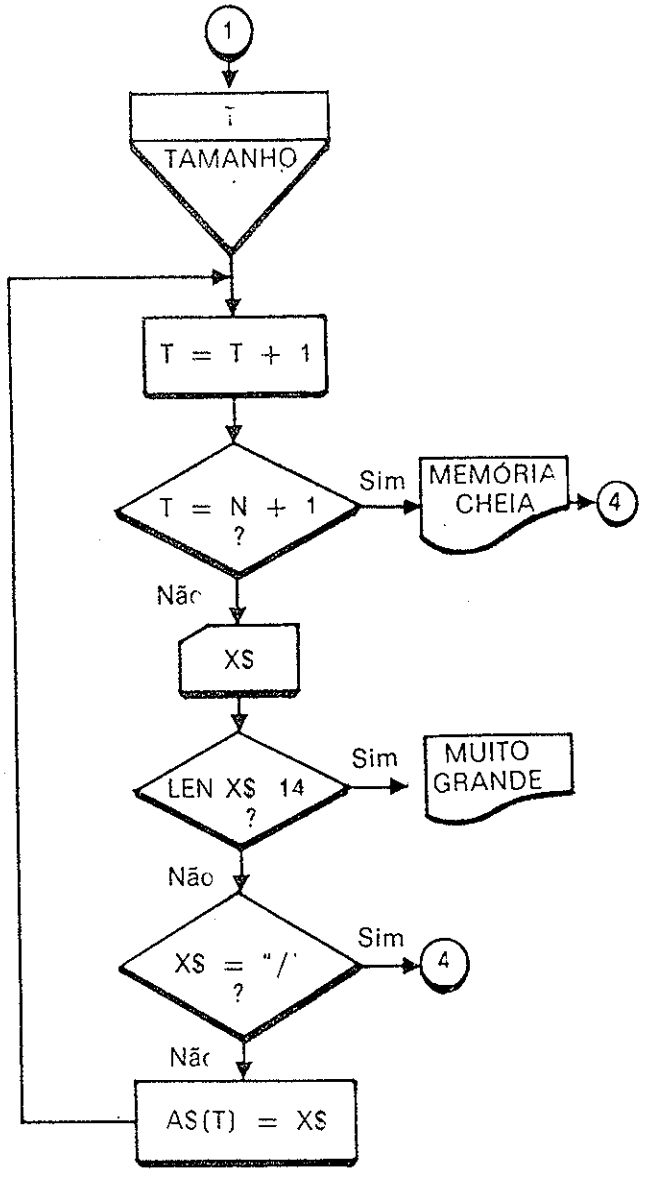
| TABELA | |
|---|---|
| I | Inserção |
| O | Ordenar |
| M | Mostrar |

**Sub-rotina que calcula o tamanho**




Continuação da Figura 7.1

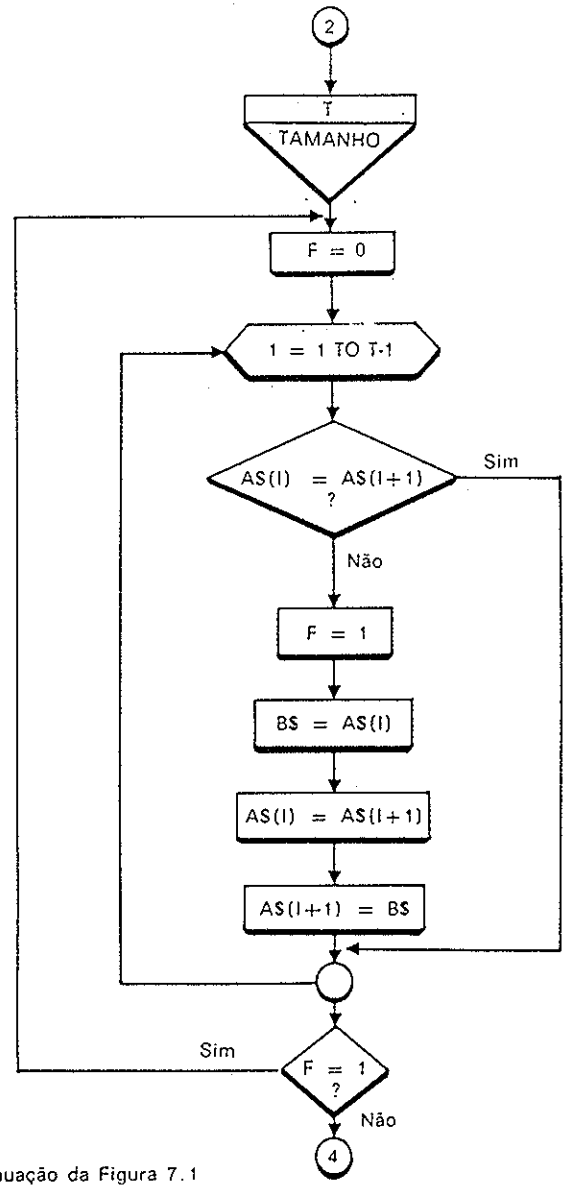
Sub-programa de Inserção




Continuação da Figura 7.1

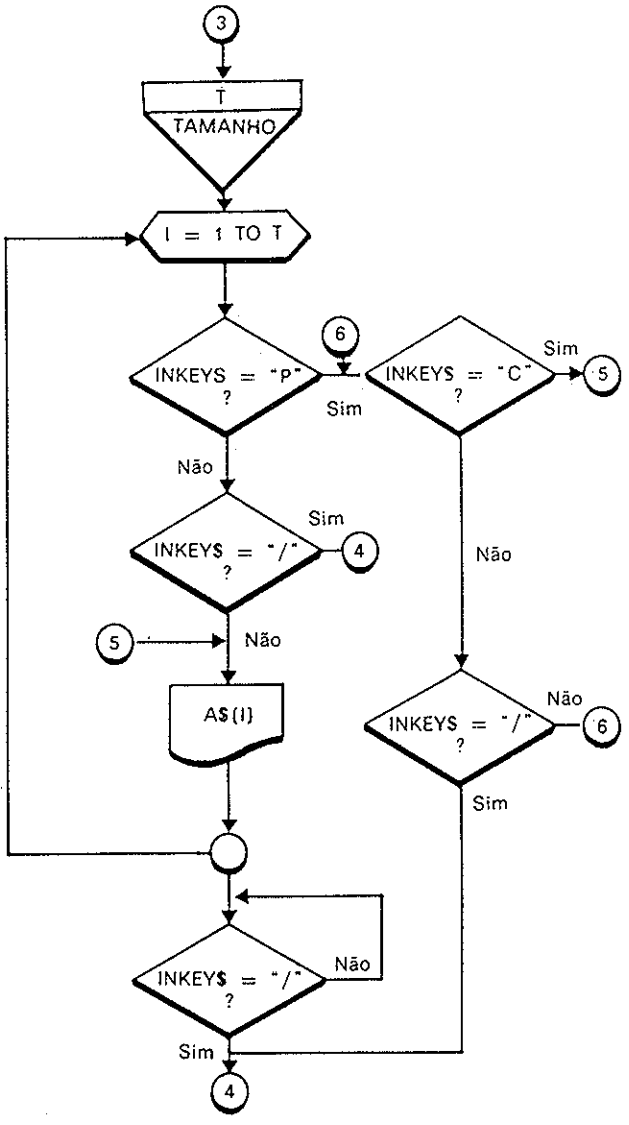


Sub-programa de Ordenação




Continuação da Figura 7.1

**Sub-programa de Mostra**




Continuação da Figura 7.1



```
  10 REM ORDEM
  20 DIM A$(25,14)
  25 SLOW
  30 PRINT TAB 4;"TABELA DE FUNC
OES"
  35 PRINT
  40 PRINT TAB 6;"I...INSERE"
  45 PRINT TAB 6;"O...ORDENA"
  50 PRINT TAB 6;"M...MOSTRA"
  80 PRINT
  85 PRINT TAB 4;"QUAL A FUNCAO
DESEJADA?"
  90 INPUT F$
  95 IF F$="I" THEN GOTO 200
 100 IF F$="O" THEN GOTO 400
 105 IF F$="M" THEN GOTO 500
 135 CLS
 140 PRINT AT 8,4;"FUNCAO NAO DE
FINIDA                FAVOR TECLAR
NOVAMENTE"
 145 PAUSE 600
 150 CLS
 155 GOTO 30
 160 FAST
 162 FOR T=1 TO 25
 164 IF A$(T)="                " T
HEN GOTO 185
 170 NEXT T
 175 LET T=25
 180 SLOW
 182 RETURN
 185 LET T=T-1
 190 GOTO 180
 200 CLS
 205 GOSUB 160
 210 LET T=T+1
 215 IF T=26 THEN GOTO 300
 220 PRINT "QUAL O ITEM ?"
 225 INPUT X$
 230 IF LEN (X$)>14 THEN GOTO 25
0
 235 IF X$="/" THEN GOTO 275
 240 LET A$(T)=X$
 241 PRINT
 242 PRINT X$
 243 PAUSE 600
 244 CLS
 245 GOTO 210
 250 PRINT
 255 PRINT "TAMANHO MAIOR QUE O
PERMITIDO:  FAVOR TECLAR NOVAMEN
TE"
```

```
260 PAUSE 500
265 CLS
270 GOTO 220
275 FAST
277 CLS
280 PRINT AT 8,4;"FUNCAO EXECUT
ADA"
285 PAUSE 500
290 CLS
295 GOTO 25
300 CLS
305 PRINT AT 8,4;"CAPACIDADE DE
MEMORIA ESGOTADA"
307 PAUSE 500
310 CLS
315 GOTO 30
400 CLS
402 GOSUB 150
404 FAST
405 LET F=0
410 FOR I=1 TO T-1
415 IF A$(I)<=A$(I+1) THEN GOTO
440
420 LET F=1
425 LET B$=A$(I)
430 LET A$(I)=A$(I+1)
435 LET A$(I+1)=B$
440 NEXT I
445 IF F=1 THEN GOTO 405
447 SLOW
450 GOTO 475
500 CLS
505 GOSUB 150
510 FOR I=1 TO T
515 IF INKEY$="P" THEN GOTO 550
520 IF INKEY$="/" THEN GOTO 275
525 SCROLL
530 PRINT A$(I)
535 NEXT I
540 IF INKEY$="/" THEN GOTO 275
545 GOTO 540
550 IF INKEY$="C" THEN GOTO 525
555 IF INKEY$="/" THEN GOTO 275
560 GOTO 550
```

O programa apresentado na figura 7.2 é razoavelmente complexo e mostra como estruturar um programa "quebrando-o".

Foram deixados "buracos" intencionais, entre os endereços, para futuras inserções. À medida em que o programa cresce, torna-se cada vez mais difícil sua compreensão. Para evitar isto, podemos inserir comentários nos pontos críticos usando a pseudo-instrução "REM".

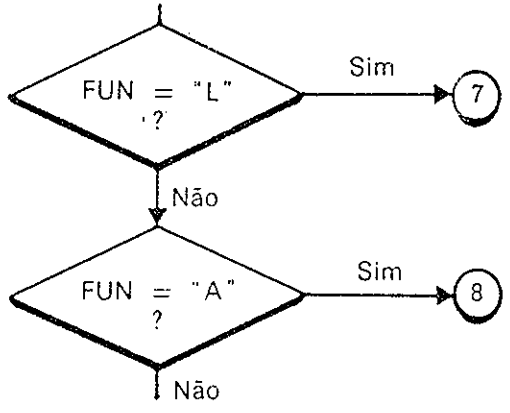


No nosso programa, seria conveniente inserir estes comentários:

```
 15 REM ESCOLHA DA FUNCAO
150 REM SUBROTINA QUE CALCULA
       O TAMANHO
190 REM INSERCAO DE DADOS
390 REM ORDENACAO DE DADOS
490 REM  MOSTRA DE DADOS
```

É conveniente colocar comentários sempre que a listagem do programa comece a se tornar complexa e desde que não hajam sérias limitações de memória.

Vamos agora introduzir as funções **limpar memória** e **achar um certo item.** Para tanto são necessárias as seguintes complementações no diagrama de blocos inicial:

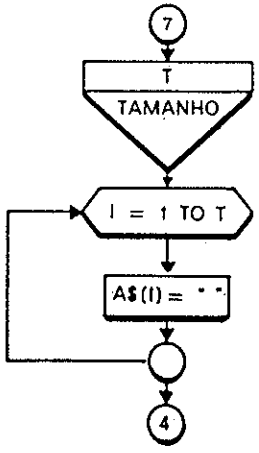



com as respectivas instruções:

```
55 PRINT TAB 6;"A...ACHA"
60 PRINT TAB 6;"L...LIMPA"
110 IF F$="L" THEN GOTO 600
115 IF F$="A" THEN GOTO 650
```

Veja agora o diagrama de blocos das duas funções:

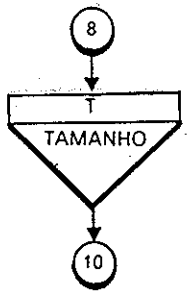
**Diagrama de blocos para LIMPAR**

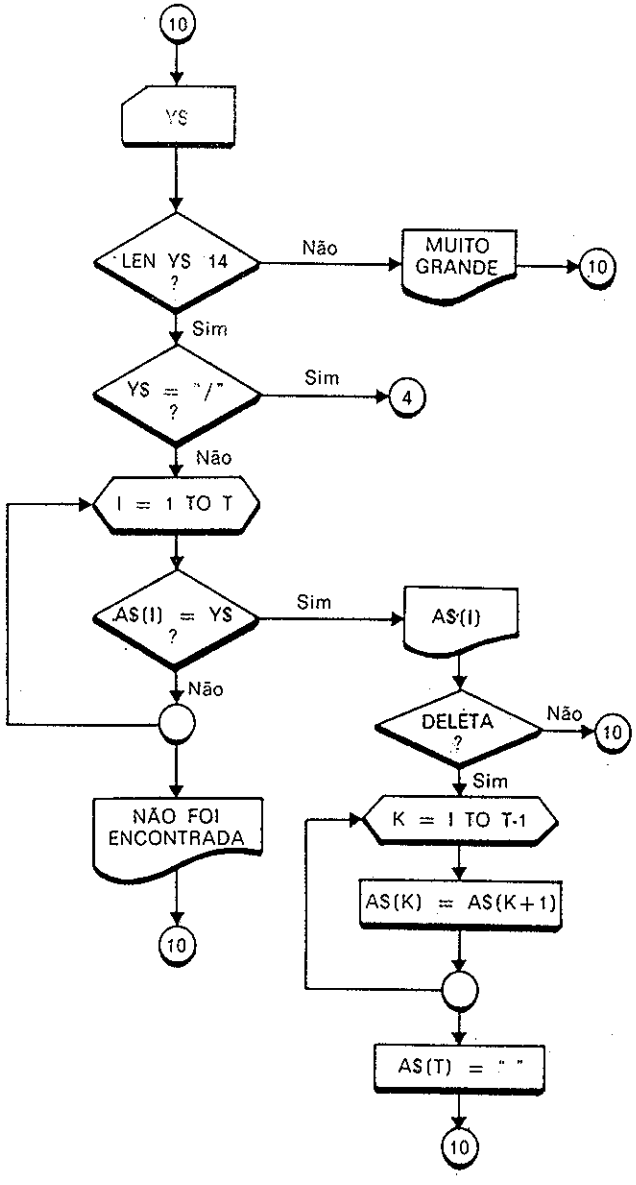



```
598 REM LIMPA
600 CLS
605 GOSUB 150
610 FAST
615 FOR I=1 TO T
620 LET A$(I)=""
625 NEXT I
630 SLOW
635 GOTO 275
```

**Diagrama de blocos para ACHAR**

(encontrando um item, vamos dar a possibilidade de retirá-lo)



```
 648 REM ACHA/DELETA
 650 CLS
 652 DIM Y$(14)
 655 GOSUB 160
 660 PRINT "QUAL O ITEM A SER EN
CONTRADO ?"
 665 INPUT P$
 670 IF LEN P$>14 THEN GOTO 795
 672 LET Y$=P$
 675 IF Y$(1)="/" THEN GOTO 275
 680 CLS
 685 FAST
 690 FOR I=1 TO T
 695 IF A$(I)=Y$ THEN GOTO 730
 700 NEXT I
 705 SLOW
 710 PRINT AT 8,4;"O ITEM NAO FO
I ENCONTRADO"
 715 PAUSE 600
 720 CLS
 725 GOTO 660
 730 SLOW
 735 PRINT AT 8,0;A$(I)
 740 PRINT
 742 PRINT "DELETA ? (S/N)"
 744 INPUT K$
 746 IF K$="S" THEN GOTO 820
 748 CLS
 750 GOTO 660
 795 PRINT
 800 PRINT "TAMANHO MAIOR QUE O
PERMITIDO  FAVOR TECLAR NOVAMEN
TE"
 805 PAUSE 600
 810 CLS
 815 GOTO 660
 820 CLS
 821 FAST
 822 FOR K=I TO T-1
 825 LET A$(K)=A$(K+1)
 830 NEXT K
 835 LET A$(T)=""
 840 SLOW
 845 GOTO 660
```

Vamos agora adicionar ao programa a "quantidade de itens" e o respectivo valor. Vamos também dar ao usuário do programa a possibilidade de "alterações" nas funções **acha** e **mostra** e de "apagamento" inclusive, nesta última.

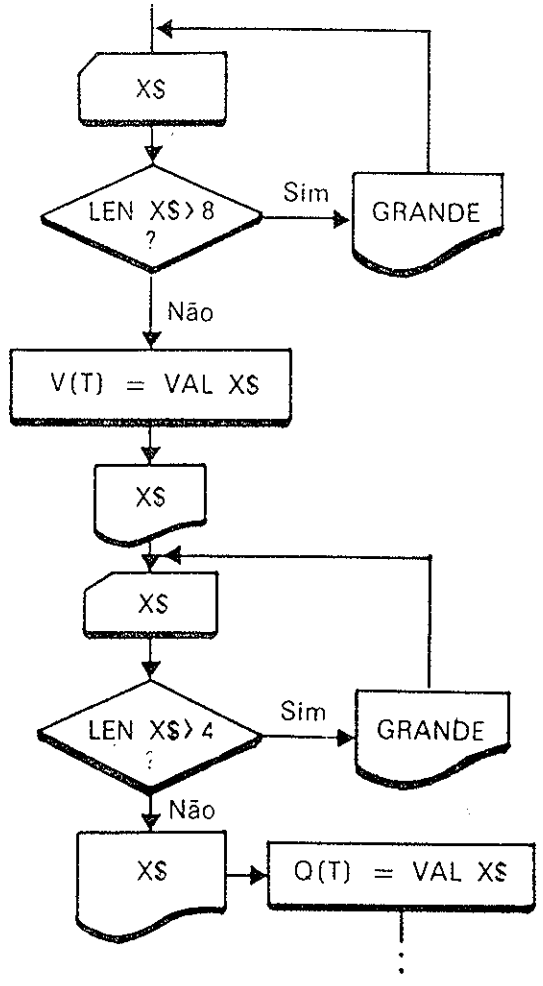


inicialmente é necessário dimensionar as matrizes de valor (V) e de quantidade (Q):

```
 16 DIM Q(25)
 18 DIM V(25)
```

Note que usaremos matrizes numéricas, pois elas costumam ocupar menos memória do que as matrizes "string".

A seguir, no sub-programa de "inserção" devemos colocar:



Note a conveniência da sub-rotina:

```
1000 PRINT
1010 PRINT "TAMANHO MAIOR QUE O
PERMITIDO.  FAVOR TECLAR NOVAMEN
TE."
1020 PAUSE 600
1030 CLS
1040 RETURN
```

Assim, as seguintes alterações são necessárias:

```
244 PRINT
245 PRINT "QUAL O VALOR?"
246 INPUT X$
247 IF LEN X$>8 THEN GOTO 268
248 LET V(T)=VAL X$
249 PRINT
250 PRINT X$
251 PRINT
252 PRINT "QUAL A QUANTIDADE?"
253 INPUT X$
254 IF LEN X$>4 THEN GOTO 272
255 LET Q(T)=VAL X$
256 PRINT
257 PRINT X$
258 PAUSE 600
259 CLS
260 GOTO 210
```

Observação: valor máximo = Cr$ 99.999.999 e quantidade máxima = 9.999

Podemos retirar a linha 243 e modificar a linha 230 para:

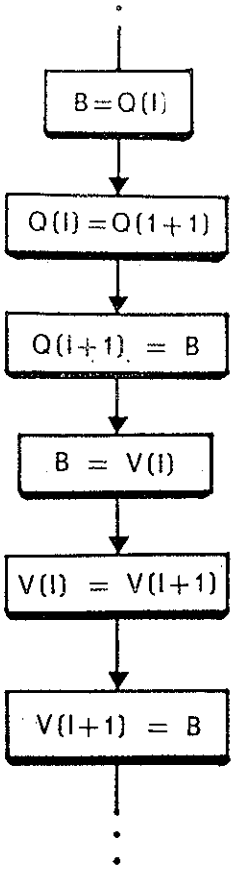
```
230 IF LEN (X$)>14 THEN GOTO 250
```

e acrescentar:

```
262 GOSUB 1000
265 GOTO 220
268 GOSUB 1000
270 GOTO 245
272 GOSUB 1000
274 GOTO 252
```

Para modificar o programa de ordenação, devemos introduzir a "troca" de quantidade e valores:



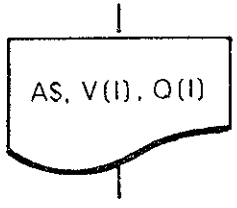



o que implica nas seguintes modificações do programa:

```
422 LET B$=A$(I)
425 LET A$(I)=A$(I+1)
427 LET A$(I+1)=B$
430 LET B=Q(I)
432 LET Q(I)=Q(I+1)
435 LET Q(I+1)=B
437 LET B=V(I)
438 LET V(I)=V(I+1)
439 LET V(I+1)=B
```

Note que à medida que implementamos o programa, novas variáveis precisam ser usadas (por exemplo: B, V(I), Q(I)). É útil portanto, ter o costume de fazer uma lista de variáveis utilizadas e as respectivas funções, para evitar a destruição de "variáveis importantes" e permitir a reutilização de "variáveis auxiliares" com a finalidade de economizar memória.

No sub-programa "mostra", a única coisa a ser alterada é a impressão:



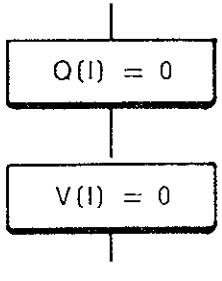

ou seja:

```
512 SCROLL
514 PRINT A$(I);TAB 15;"CR$ ";V
(I);TAB 28;Q(I)
```

e eliminar as linhas 525 e 530!

Esta inversão, ou seja, a colocação da impressão antes dos testes de INKEYS, é necessária para que, no caso de interrupção com INKEYS = "P", a variável I esteja indicando o valor da posição do último item mostrado na tela. Isto para poder comandar convenientemente as funções que serão introduzidas para "modificação" e "apagamento".

Resta ainda modificar **limpa** e **acha.** Assim, na função **limpa**, devemos acrescentar:

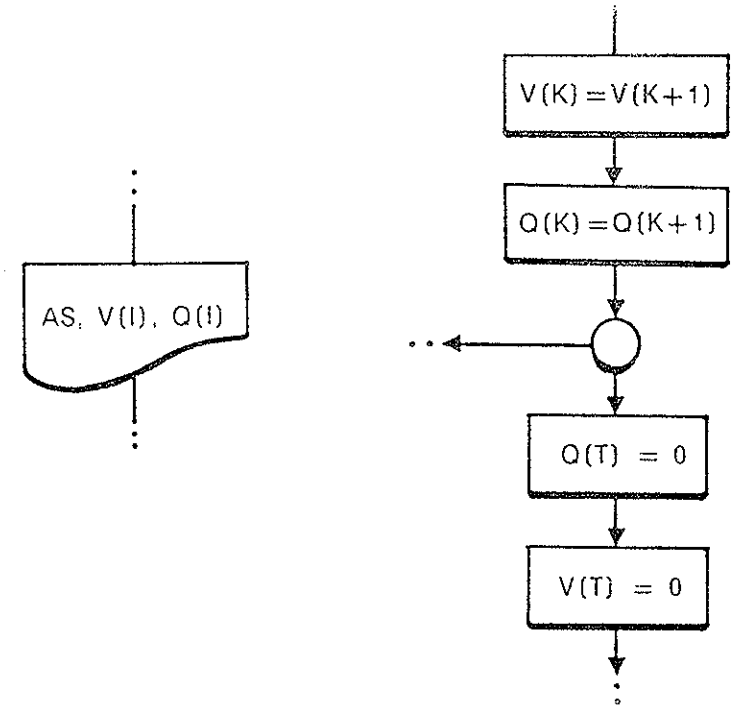





que equivale a:

```
622 LET Q(I)=0
624 LET V(I)=0
```

e na função **acha:**

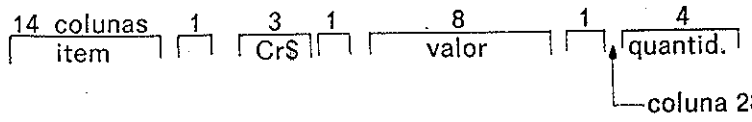



o que corresponde a:

```
735 PRINT AT 8,0;A$(I);TAB 15;"
CR$ ";V$(I);TAB 29,0;Q(I)
826 LET V(K)=V(K+1)
827 LET Q(K)=Q(K+1)
836 LET Q(T)=0
837 LET V(T)=0
```

Vamos "rodar" o programa. Pode-se notar a inconveniência deste sistema de impressão para números, pois eles ficam encostados à esquerda. Vamos ver como podemos fazer para

encostá-los à direita. O nosso formato de impressão é:




Pode-se concluir que as instruções para "impressão" devem ser alteradas para:

```
 514 PRINT A$(I);TAB 15;"CR$ ";T
AB (27-LEN (STR$ V(I)));V(I);TAB
 (32-LEN (STR$ Q(I)));Q(I)
```

Idem para a linha 735. Cuidado! Não precisa digitar tudo outra vez! Basta trazer para baixo, com EDIT, a linha 514, apagar sua etiqueta e escrever 735 no lugar.

```
 735 PRINT A$(I);TAB 15;"CR$ ";T
AB (27-LEN (STR$ V(I)));V(I);TAB
 (32-LEN (STR$ Q(I)));Q(I)
```

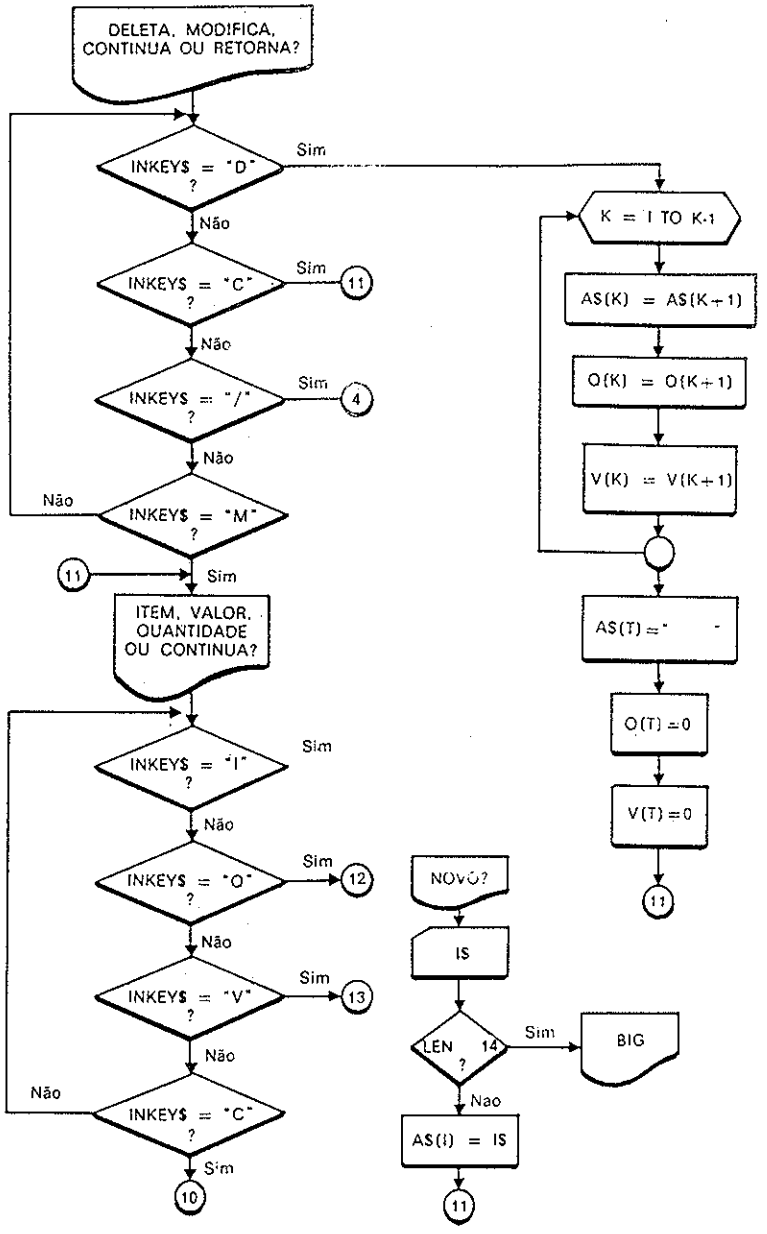
Vamos rodar novamente o programa, porém, apertando GO-TO 25 no lugar de RUN. Selecione a função **mostre.** O que acontece?

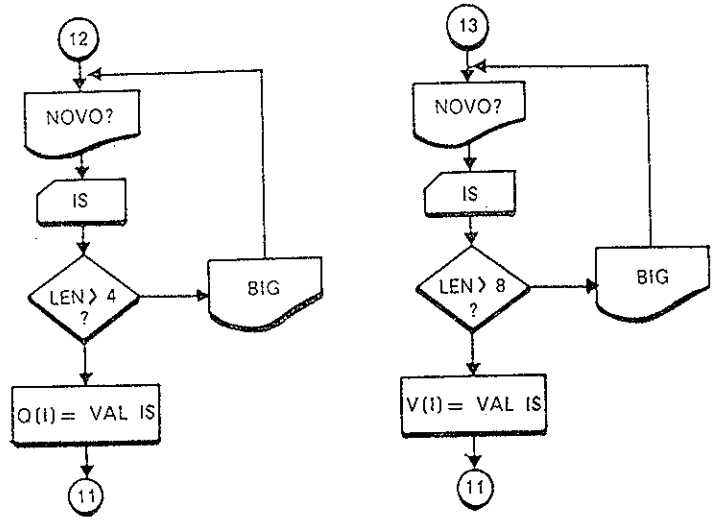
```
RUN

GOTO 25
```

Iremos agora dar a possibilidade de alterar o item, valor ou quantidade na função **acha:**








o que implica em:

```
 741 PRINT "DELETA,MODIFICA,CONT
INUA OU      RETORNA?    (D,M,C,/)
"
 742 IF INKEY$="D" THEN GOTO 820
 743 IF INKEY$="C" THEN GOTO 747
 744 IF INKEY$="/" THEN GOTO 275
 745 IF INKEY$="M" THEN GOTO 749
 746 GOTO 742
 747 CLS
 748 GOTO 660
 749 PRINT
 750 PRINT "ITEM,VALOR,QUANTIDAD
E OU CONTI -NUA ? (I,V,Q,C)"
 751 IF INKEY$="I" THEN GOTO 756
 752 IF INKEY$="V" THEN GOTO 767
 753 IF INKEY$="Q" THEN GOTO 778
 754 IF INKEY$="C" THEN GOTO 747
 755 GOTO 751
 756 PRINT
 757 PRINT "QUAL O NOVO ITEM ?"
 758 INPUT I$
 759 IF LEN I$>14 THEN GOTO 765
 760 LET A$(I)=I$
 761 PRINT I$
```



```
 762 PAUSE 600
 763 CLS
 764 GOTO 750
 765 GOSUB 1000
 766 GOTO 756
 767 PRINT
 768 PRINT "QUAL O NOVO VALOR ?"
 769 INPUT I$
 770 IF LEN I$>8 THEN GOTO 776
 771 LET V(I)=VAL I$
 772 PRINT I$
 773 PAUSE 600
 774 CLS
 775 GOTO 750
 776 GOSUB 1000
 777 GOSUB 767
 778 PRINT
 779 PRINT "QUAL A NOVA QUANTIDA
DE ?"
 780 INPUT I$
 781 IF LEN I$>4 THEN GOTO 787
 782 LET Q(I)=VAL I$
 783 PRINT I$
 784 PAUSE 600
 785 CLS
 786 GOTO 750
 787 GOSUB 1000
 788 GOTO 778
 800 GOTO 660
```

Podemos, então, retirar as linhas: 805, 810 e 815

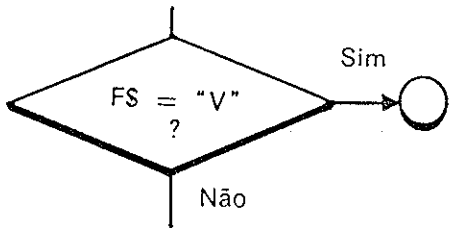
Como última implementação, vamos possibilitar o cálculo do "valor médio" do estoque. As demais alterações serão feitas em exercícios.

O **valor médio** é calculado pela relação:

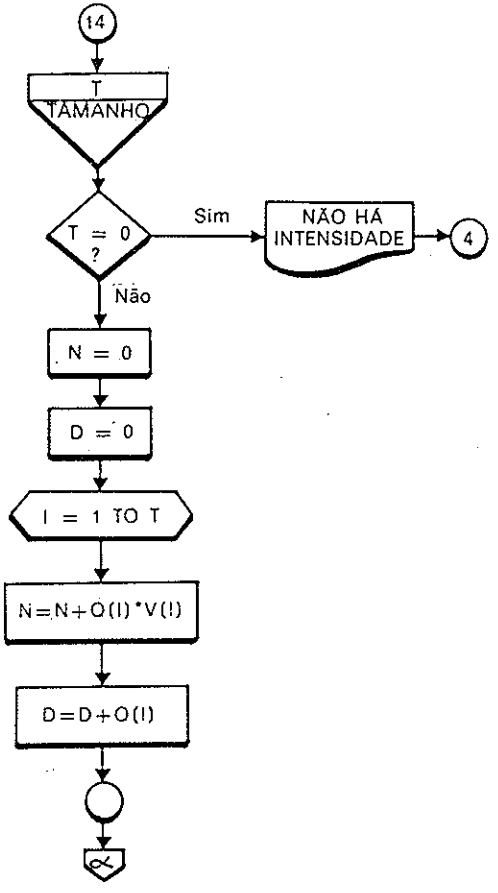
$$VM = \frac{\sum_{x=1}^{T} Q(I) \cdot V(I)}{\sum_{I=1}^{T} Q(I)}$$

Assim, iremos definir a função:

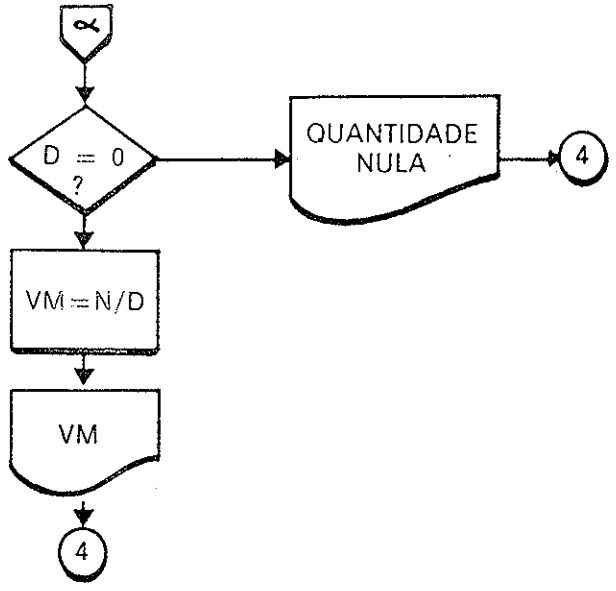
V ... Valor médio




```
 55 PRINT TAB 5;"V...VALOR MEDIO"
120 IF F$="V" THEN GOTO 900
```






```
900 CLS
905 GOSUB 160
907 IF T=0 THEN GOTO 962
910 FAST
915 LET N=0
920 LET D=0
925 FOR I=1 TO T
930 LET N=N+Q(I)*V(Z)
935 LET D=D+Q(I)
940 NEXT I
942 IF D=0 THEN GOTO 966
945 LET VM=INT (N/D)
950 PRINT AT 8,0;"VALOR MEDIO="
;VM
955 PAUSE 600
960 GOTO 275
962 PRINT AT 8,4;"NAO HA ITENS"
963 PAUSE 600
964 GOTO 275
966 PRINT AT 8,2;"QUANTIDADE DE
ITENS NULA"
968 PAUSE 600
970 GOTO 275
```

e vamos aproveitar para inserir a função **mostra:**

```
507 IF T=0 THEN GOTO 962
```

## COMENTÁRIOS

Em termos de "funções básicas" o programa está completo. Obviamente há muitos outros detalhes que podemos implementar, alguns dos quais são sugeridos nos exercícios.

Note a estrutura dos raciocínios que foi seguida:

1) programa "fechado em si mesmo";

2) programa imune a qualquer erro de entrada, pois deve poder ser usado por qualquer pessoa;

3) estrutura de programação:

- procurar definir sub-rotinas convenientes;
- quebrar em sub-programas para facilitar o raciocínio;
- deixar "buracos" razoáveis para futuras implementações;
- fazer uma lista de variáveis especificando suas respectivas funções;
- raciocinar sempre em diagrama de blocos.

## EXERCÍCIOS

1) Uma das sugestões feitas na teoria não foi efetuada: a possibilidade de alterar ou deletar dados na função **mostra** após "parar a tela". Como exercício procure introduzir esta modificação.

Obviamente o programa deve testar se o bloco "cabe" ou seja, se o conjunto todo não atinge a última posição de memória reservada.

2) Os seguintes comandos poderiam ser introduzidos no programa anterior:

a) procurar e listar todos os itens que comecem com uma dada letra;

b) procurar e listar todos os itens cujo valor esteja entre dois valores dados.

Modifique o programa para que ele aceite estes comandos.



3) Ainda com relação a este programa, podemos notar que ele é "fechado em si mesmo". O objetivo do mesmo é que ele possa ser usado por uma pessoa qualquer e que teste **todas** as possibilidades de erro para evitar que o programa seja interrompido. Assim, por exemplo, testamos se o tamanho das palavras dos itens é maior do que 14. Note que quando introduzimos o **valor** e a **quantidade**, se teclarmos um caractere não numérico, o programa parará acusando erro:

2/248 (no valor)

ou

2/255 (na quantidade)

Invente uma proteção que após as perguntas:

QUAL O VALOR?

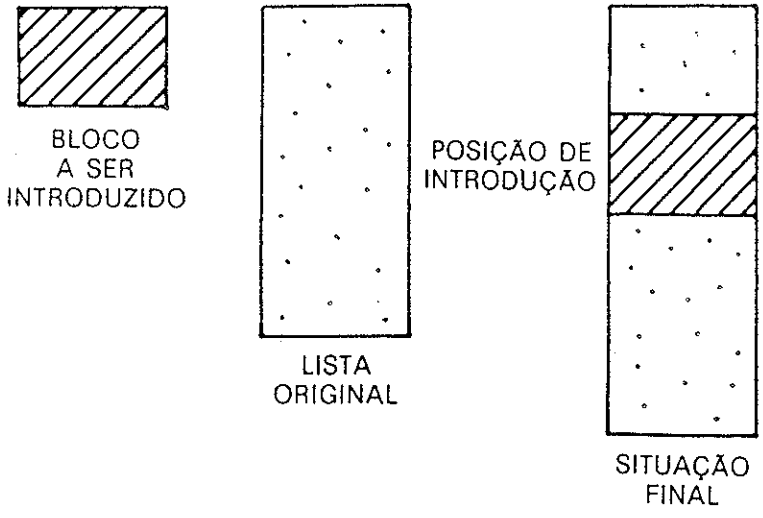
e

QUAL QUANTIDADE?

faça com que elas sejam repetidas em caso de teclagem errada.

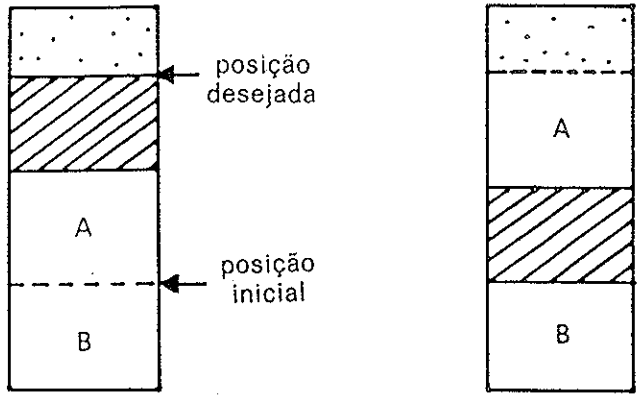
Sugestão: teste o código de cada caractere introduzido, pois os códigos dos números estão na faixa de 28 a 37.

4) Faça um programa que, dada uma lista de itens, seja capaz de introduzir em uma dada posição um novo bloco de itens:

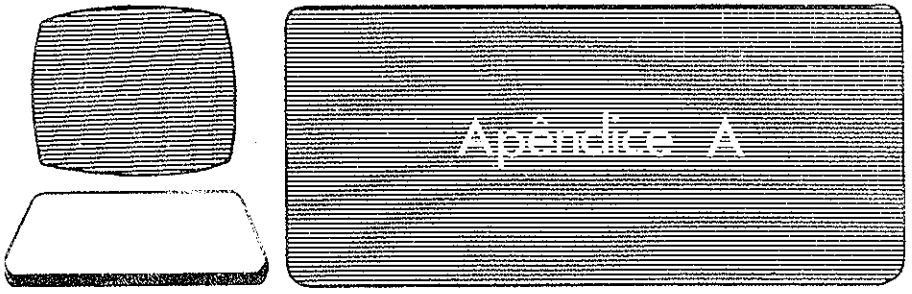


5) Complementando o programa anterior, faça com que seja possível "deletar" um bloco de itens da lista.

6) Faça agora com que seja possível "deslocar" um bloco de itens de uma posição para outra.






Neste apêndice apresentamos um resumo de todos os caracteres (comandos, símbolos gráficos, etc...) dos micros da linha SINCLAIR (TK82/83/85, CP200, etc...).

O número antes da explicação do caractere corresponde ao código do caractere indicado. Quando existirem dois números referentes a um mesmo caractere, o primeiro indica o código do caractere normal e o segundo, o código de caractere inverso.

**ABS**
210 Fornece o valor absoluto (módulo) de seu argumento (ex.: ABS −7=7).

**ACS (ARCCOS)**
203 Fornece o ângulo em radianos quando seu argumento é o cosseno desse ângulo (−1 argumento 1).

**AND**
218 Operador lógico: combina duas expressões obrigando a ambas serem verdadeiras para que a combinação também seja.

**ASN (ARCSIN)**
202 Fornece o ângulo em radianos quando seu argumento é o seno desse ângulo (−1< = argumento< =1).

**AT**
193 É uma função de impressão utilizável com PRINT.

PRINT AT X,Y; "imprime" na linha X (0 a 21) e na coluna Y (0 a 31).

**ATN (ARCTAN)**
204 Fornece o ângulo em radianos quando seu argumento é a tangente desse ângulo (−1 argumento 1).

**BREAK**
BREAK interrompe a execução do programa com o código D/n onde n é a linha em que ocorreu a interrupção.

**CHR$**
214 CHR$ n (para 0 n 255) fornece o caracter cujo código é n.

## CLEAR
253 Apaga todas as variáveis tornando-as indefinidas. As linhas do programa nada sofrem.

## CLS
251 Esta instrução limpa completamente a tela (limpa a memória de vídeo: DISPLAY FILE).

## CODE
196 Fornece o código de um caracter, do primeiro caracter de uma "string" ou do caracter solicitado de uma "string" (ex.: CODE X$(2)=39 se X$="ABC").

## CONT
232 Esta instrução faz o programa continuar sua execução após ter parado por STOP, BREAK, ou falta de espaço na tela (códigos, respectivamente, 9/ , D/ e 5/ ).

## COPY
255 COPY é um comando que faz a impressora tirar cópia exata da tela.

## COS
200 Fornece o cosseno de um ângulo expresso em radianos.

## DIM
233 Reserva o espaço na memória para o uso de variáveis subscritas (matrizes). Todos os espaços reservados são inicialmente "zerados" (ex.: DIM A (4,4) ou DIM B$ (5,2)).

## EDIT
117 Quando pressionada juntamente com a tecla SHIFT (cursos K ou L ), "desce" a linha para a qual aponta o cursor de edição.

## EXP
206 EXP X calcula $e^x$ onde e= 2,7182818.

## FAST
229 Modo de operação 4 vezes mais rápido que o SLOW pois o processamento é feito sem interrupções. A tela só é mostrada em INPUT e PAUSE.

## FOR
235 Define a variável de controle de um "looping" (ex.: FOR N=0 TO 100 ou FOR H=1 TO 4 STEP .1 ou FOR J=10 TO 1 STEP −2).

Deve ser sempre "fechado" por um NEXT.

## FUNCTION
121 Pressionada simultaneamente com a tecla SHIFT muda o cursor para o modo F para permitir a entrada de uma função.

## GOSUB
237 GOSUB n desvia o programa para a execução da sub-rotina que se inicia na linha n e termina com um RETURN. Após o RETURN, o programa volta à linha seguinte ao GOSUB que deu origem ao desvio.

## GOTO
236 GOTO n desvia ou simplesmente remete a execução de um programa para a linha n.



## GRAPHICS
116 Pressionada juntamente com SHIFT, muda o cursor L ou K para a G e vice-versa.

## IF
250 É a instrução que permite uma tomada de decisão. Deve ser utilizada na forma: IF (condição) THEN (instrução).

## INKEY$
65 Lê o caracter (modo L ) da tecla pressionada. Se não for pressionada qualquer tecla, retorna a "string" vazia ("").

## INPUT
238 INPUT interrompe a execução de um programa para permitir a entrada de dados. O seu argumento deve ser uma variável numérica, alfanumérica ("string") ou subscrita previamente dimensionada por uma instrução DIM.

## INT
207 Fornece a parte inteira do número utilizado como argumento arredondando para menos.

## LEN
198 Fornece o comprimento de uma "string" (ex.: LEN "ABC"=3).

## LET
241 É uma instrução usada para definir uma variável numérica, "string", indexadas ou não (ex.: LET A=5 significa "faça" A=5).

## LIST
240 LIST é um comando que "imprime" no vídeo a listagem de um programa a partir da linha "zero". LIST n "imprime" a listagem na tela a partir da linha n.

## LN
205 Fornece o logaritmo neperiano (na base e) de um número 0.

## LOAD
239 LOAD "" ou LOAD "nome do programa" carregam o computador com um programa a variáveis que estão sendo transmitidos a 300 bits por segundo por um gravador.

## LLIST
226 Funciona da mesma forma que LIST, sendo que a listagem é obtida na impressora.

## LPRINT
225 Análogo à PRINT com a diferença que a impressão é efetuada na impressora. Permite o uso da vírgula, ponto e vírgula e TAB.

## NEW
230 Este comando apaga o programa, as variáveis e a tela deixando a RAM "limpa" até o endereço da "RAM-TOP".

## NEW LINE
118 É o comando que remete ao computador uma instrução, uma linha do programa ou dados de um INPUT. É a porta de entrada do sistema.

Em qualquer caso o computador verifica a validade do coman-

do. Se algum erro de sintaxe ocorrer deve aparecer o cursor S próximo ao erro. Outros erros são comunicados com um código de reportagem.

Tecla auxiliar que deve ser pressionada simultaneamente com outra para se obterem as várias funções dessa última.

## NEXT
243 Esta instrução fecha o "looping" FOR-NEXT. Deve ter como argumento a mesma variável definida pelo comando FOR correspondente.

## NOT
215 Operador lógico de negação (ex.: NO é o mesmo que =).

## OR
217 Operador lógico: combina duas expressões obrigando a apenas uma delas ser verdadeira para que a combinação também seja.

## PAUSE
242 PAUSE n faz o computador parar a execução de um programa por n/60 segundos se n 32767 e indefinidamente se 32767 n 65535.

Se durante a pausa alguma tecla for pressionada, o computador retoma a execução do programa.

## PEEK
211 PEEK n (para S n 65535) é igual ao byte contido no endereço n da memória.

## PI (π)
66 É a constante 3,141592653. No vídeo é impressa como PI.

## PLOT
246 PLOT, X,Y faz com que a posição X,Y da tela fique preta. Na horizontal, X varia de 0 a 63 e na vertical, Y varia de 0 a 43. A posição 0,0 fica no canto inferior esquerdo da tela.

## POKE
244 POKE X,Y aloca diretamente no endereço X (0 a 655535) o valor do byte Y (−255 a 255).

## PRINT
245 É o comando de "impressão" na tela. Tem vários efeitos:
- *PRINT imprime" uma linha em branco.
- *PRINT X ou X$ "imprime" variáveis.
- *PRINT "ABC" "imprime" uma cadeia de caracteres.
- *PRINT AT X,Y; "imprime" na linha X (0 a 21) e na coluna Y (0 a 31).
- *PRINT TAB X; "imprime" na coluna X.

## RAND
249 RAND faz a seqüência aleatória da função RND ter um valor inicial também aleatório. RAND n (onde, 0 n 65535) faz a seqüência ter um valor inicial determinado para cada valor de n.

## REM
234 REM é abreviatura de "remarks" (observações). Não tem qualquer efeito e pode ser seguido de quaisquer caracteres. É muito usado para dar títulos a sub-rotinas de programas e para guardar programas escritos em linguagem de máquina.



## RETURN
254 É a instrução de término de uma sub-rotina. Não tem argumento e deve ter um GOSUB correspondente.

Provoca o retorno à linha seguinte ao GOSUB que causou o desvio.

## RND
64 Gera uma seqüência pseudo-aleatória de números entre 0 e 1. Não é exatamente aleatória pois está pré-definida por RAND.

## RUBOUT
119 Pressionada juntamente com SHIFT, causa o apagamento do caracter à esquerda do cursor.

## RUN
247 A instrução RUN apaga as variáveis e roda o programa a partir da linha 0. RUN n roda o programa a partir da linha n.

## SCROLL
231 Esta instrução desloca a tela toda de uma linha para cima. A linha superior se perde e a inferior fica vazia. A posição de impressão passa a ser a do início da última linha após um SCROLL.

## SAVE
248 Esta instrução envia para o gravador o programa e as variáveis a uma velocidade de 300 bits por segundo. Deve ter como argumento uma "string" não vazia.

## SGN
209 Fornece o sinal de um número retornando −1,0 ou 1 se o argumento for respectivamente negativo, nulo ou positivo.

## SIN
199 Fornece o seno de um ângulo expresso em radianos.

## SLOW
228 Modo de operação do computador em que o processamento é feito entre cada imagem de vídeo. É mais lento e a tela é continuamente apresentada.

## SQR
208 SQR X extrai a raiz quadrada de X 0.

## STEP
224 Usado nos "loopings" FOR-NEXT. Permite incrementos ou decrementos na variável de controle diferentes de 1 (ex.: FOR I=1 TO 10 STEP 2).

## STOP
227 Pára a execução de um programa com o código de reportagem 9/n onde n é a linha que contém o STOP. Comandando CONT o programa continua a sua execução na próxima linha à n.

## STR$
213 Transforma uma variável numérica em "string" (ex.: STR$ 5= "5").

## TAB
194 TAB X desloca a posição de impressão para a coluna X da primeira linha livre. É usada com PRINT ou LPRINT.

## TAN
201 Fornece a tangente de um ângulo expresso em radianos.

## THEN
222 Deve ser usada em conjunto com IF (IF condição THEN instrução). Muda o cursor L para K

## TO
223 Usado em "loopings" FOR-NEXT (ex.: FOR A=1 TO 10) e em "slicings" (ex.: PRINT AS(3 TO 7)).

## UNPLOT
252 UNPLOT X,Y "apaga" a posição X,Y da tela. Tem funcionamento análogo a PLOT X,Y.

## USR
212 É uma função que chama sub-rotinas em linguagem de máquina. O seu argumento é o endereço inicial da sub-rotina. Deve ser usada com alguma "palavra-chave" (ex.: RAND USR n).

## VAL
197 Calcula o valor numérico de uma "string" (ex.: VAL "1+2"=3 ou VAL "9**2"=81 ou VAL FS=8 se FS="X*2" e X=4).

"
11 139 Deve ser usado para abrir e fechar cadeias de caracteres de variáveis "string" ou cadeias a serem impressas.

$
13 141 É o símbolo que deve ser acrescentado a uma letra para que esta possa denotar uma variável "string" (alfanumérica).

:
14 142 Dois pontos: não tem qualquer função sintática.

?
15 143 Ponto de interrogação: não tem qualquer função sintática.

(
16 144 Usado como inversor de prioridades nas operações aritméticas. Também utilizado no dimensionamento e definição de variáveis subscritas.

)
17 145 Deve sempre existir um ) para cada ( utilizado anteriormente ou haverá erro de sintaxe.

>
18 146 Sinal de "maior" utilizado na comparação lógica entre duas variáveis.

<
19 147 Sinal de "menor" utilizado na comparação lógica entre duas variáveis.

=
20 148 Sinal de igual.

+
21 149 Operador da adição numérica ou de "strings" (ex.: AS+BS).

-
22 150 Operador da subtração ou atribuição do "menos" unário (ex.: LET A=-2).

*
23 151 É o operador do produto.



/
24 152 É o operador da divisão.

;
25 153 Ponto e vírgula: usado para encadear várias impressões numa mesma instrução PRINT (ex.: PRINT AT 1,0;"A"; TAB 5;"E";X;). Quando uma instrução PRINT termina com ; a próxima impressão será feita a seguir dessa.

,
26 154 A vírgula desempenha várias funções sintáticas:
*PRINT AT 3,4; "ABC"
*PRINT "A" ,, "B" (cada vírgula desloca a posição de impressão meia tela à frente)
*DIM A(4,2,3)
*LET AS(4,1)=5
*PLOT X,Y ou UNPLOT X,Y
*POKE A,B
Esta função só funciona como comando direto durante a execução de um programa. Não funciona em modo de edição ou durante um INPUT.

.
27 155 O ponto em vídeo normal é utilizado em notação decimal de valores numéricos (ex.: 5,75).

""
192 Corresponde a " que deve ser impressa no meio de uma cadeia de caracteres de uma instrução PRINT.

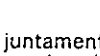
**
216 Elevação a uma potência (X**Y equivale a XY. A base X não pode ser negativa.

<=
219 Sinal de "menor ou igual" utilizado na comparação lógica de duas variáveis.

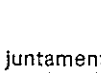
>=
220 Sinal de "maior ou igual" utilizado na comparação lógica de duas variáveis.

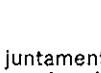
<>
221 Sinal de "diferente" utilizado na comparação lógica de duas variáveis.

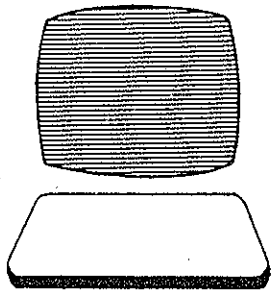
□ ou ■
0 128 Espaço (cursor K ou L ) ou quadrado negro (cursor G ).


112 Pressionada juntamente com SHIFT, move o cursor de edição para a linha de cima.


113 Pressionada juntamente com SHIFT, move o cursor de edição para a linha de baixo.


114 Pressionada juntamente com SHIFT, move o cursor de edição para a esquerda.

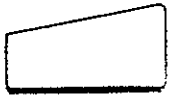

115 Pressionada juntamente com SHIFT, move o cursor de edição para a direita.



# Apêndice B

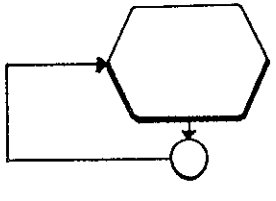
Neste apêndice apresentamos um resumo dos símbolos utilizados nos fluxogramas.

1) **Entrada de dados por teclado:**
(BASIC: INPUT)



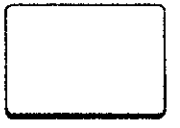
2) **Saída de dados para a tela:**
(BASIC: PRINT)

3) **Loops:**
(BASIC: FOR/NEXT)





4) **Operações e funções:**
(BASIC: LET/CLS/RAND/+/− ...)



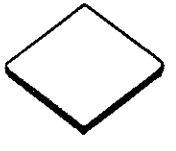
5) **Início de programa:**


6) **Fim de programa:**

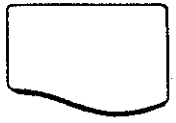

7) **Continuação do fluxograma, quando não há espaço físico:**


* qualquer símbolo, exceto I, F ou R

8) **Condicional:**
(BASIC: IF)

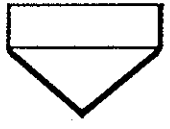


9) **Saída pela impressora:**
(BASIC: PRINT)



10) **Sub-rotinas:**
(BASIC: GOSUB)

I) chamada de sub-rotina



II) início da sub-rotina